UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

NATHÁLIA DE MENESES ALVES

Uma Abordagem Não Intrusiva e Automática para Configuração do Hadoop

Maceió
2015

NATHÁLIA DE MENESES ALVES

Uma Abordagem Não Intrusiva e Automática para Configuração do Hadoop

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal de Alagoas.

Orientador: Prof. Dr. André Lage Freitas
Coorientador: Prof. Dr. Aydano Pamponet Machado

Maceió
2015

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária Responsável: Helena Cristina Pimentel do Vale

A474u Alves, Nathalia de Meneses.
Uma abordagem não intrusiva e automática para configuração do HADOOP / Nathalia de Meneses Alves. - 2015.
101 f. : il.

Orientador: André Lage Freitas.
Coorientador: Aydano Pamponet Machado.
Dissertação (mestrado em Informática) – Universidade Federal de Alagoas. Instituto de Computação. Programa de Pós-graduação em Informática. Maceió, 2015.

Bibliografia: f. 77-80.
Apêndices: 81-101.

1. Configuração automática. 2. MapReduce. 3. HADOOP. 4. Grandes quantidades de dados. 5. Computação em nuvem. 5. Algoritmos genéticos. 6. Aprendizado em máquina. I. Título.

CDU: 004.89

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Informática – PpgI
Instituto de Computação
Campus A. C. Simões BR 104-Norte Km 14 BL 12 Tabuleiro do Martins
Maceió/AL - Brasil CEP: 57.072-970 I Telefone: (082) 3214-1401

Membros da Comissão Julgadora da Dissertação de Mestrado de Nathália de Meneses Alves, intitulada: "*Uma Abordagem Não Intrusiva e Automática para Configuração do Hadoop*", apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas em 29 de setembro de 2015, às 16h00min, no Miniauditório do Instituto de Computação da UFAL.

COMISSÃO JULGADORA

Prof. Dr. André Lage Freitas
UFAL – Instituto de Computação
Orientador

Prof. Dr. Aydano Pamponet Machado
UFAL – Instituto de Computação
Coorientador

Prof. Dr. Heitor Soares Ramos Filho
UFAL – Instituto de Computação
Examinador

Prof. Dr. Francisco Vilar Brasileiro
UFCG – Universidade Federal de Campina Grande
Examinador

A Deus, minha fonte de paciência e persistência.
A minha família pelo amor e dedicação.

AGRADECIMENTOS

Agradeço a Deus pela força e coragem para conseguir subir mais esse degrau na minha vida profissional, e muito além do profissional.

Agradecer a minha mãe Ivanilda e meu pai Manoel pela dedicação e compreensão incondicional. Amo vocês sem limites de tempo e espaço.

Agradecer a minha irmã e melhor amiga Nayara que sempre foi uma inspiração, que me apoiou e me deu força em todas as situações (boas e nem tão boas assim). Irmã, você para mim é a maior referência de profissional que eu tenho: uma pessoa que faz o que nasceu pra fazer. Espero um dia ser como você.

Agradecer aos meus tios queridos Cristina e Ricardo pelo apoio e pelas conversas que sempre me estimularam a não desistir.

Dedico essa vitória também aos meus amigos André, Thayse e Suzy que sempre acompanharam bem de perto toda a luta e torcem pelas minhas conquistas sempre. Amo muito vocês.

Dedico, especialmente, esse trabalho a minha avó Severina que não pôde ver essa etapa concluída. Sempre foi de poucas palavras, mas sempre me estimulou a não parar mesmo sem entender direito o que é ser mestre. Ela sempre será minha referência de amor e perseverança. A gente se encontra um dia, Voinha.

Agradeço também aos colegas Izadora, Israel e Yves pela ajuda para concretizar essa trabalho.

Agradeço ao professor Aydano Machado pelas preciosas ajudas nesse trabalho. Sua presença no estudo foi de suma importância e sou muito grata por isso.

Dedico também a conclusão desse trabalho ao professor André Lage pela paciência e pelo tempo investido no meu estudo. Sou muito grata a ele e espero que a minha vida profissional seja reflexo da sua competência e inteligência.

"Não importa se só tocam o primeiro verso da canção, a gente escreve o resto sem muita pressa, com muita precisão. Nos interessa o que não foi impresso e continua sendo escrito à mão. Escrito à luz de velas, quase na escuridão, longe da multidão. Somos um exército, o exército de um homem só. No difícil exercício de viver em paz."

Humberto Gessinger

RESUMO

Nas últimas décadas, a quantidade de dados gerados no mundo tem aumentado de maneira significativa. A Computação em Nuvem juntamente com o modelo de programação MapReduce, através do arcabouço Hadoop, têm sido utilizados para o processamento desses dados. Contudo, os sistemas contemporâneos ainda são complexos e dinâmicos, tornando-se difíceis de se configurar. A configuração automática de software é uma solução para esse problema, ajudando os programadores e administradores gerir a complexidade desses sistemas. Por exemplo, há soluções na literatura que utilizam aprendizado de máquina para a configuração automática do Hadoop com o intuito de melhorar o desempenho das suas aplicações. Apesar desses avanços, as soluções atuais para configurar automaticamente o Hadoop utilizam soluções muito específicas, aplicando algoritmos de aprendizagem de máquinas isoladamente. Assim, esses algoritmos não são comparados entre si para entender qual abordagem é mais adequada para a configuração automática do Hadoop. Além disso, essas soluções são intrusivas, ou seja, expõem detalhes operacionais para programadores e/ou administradores de sistemas. Esse trabalho tem por objetivo propor uma abordagem transparente, modular e híbrida para melhorar o desempenho de aplicações Hadoop. A abordagem propõe uma arquitetura e implementação de software transparente que configura automaticamente o Hadoop. Além disso, a abordagem propõe uma solução híbrida que combina Algoritmos Genéticos e várias técnicas de aprendizado de máquina (*machine learning*) implementadas em módulos separados. Um protótipo de pesquisa foi implementado a avaliado mostrando que a abordagem proposta consegue diminuir significativamente o tempo de execução das aplicações Hadoop WordCount e Terasort. Além disso, a abordagem consegue convergir rapidamente para a configuração mais adequada de cada aplicação, alcançando baixos níveis de custos adicionais (*overhead*).

Palavras-chaves: Configuração automática. MapReduce. Hadoop. Grandes quantidade de dados. Computação em Nuvem. Algoritmos Genéticos. Aprendizado de máquina.

ABSTRACT

The amount of digital data produce in the last years has increased significantly. MapReduce framework such as Hadoop have been widely used for processing big data on top of cloud resources. In spite of these advances, contemporary systems are complex and dynamic which makes them hard to configure in order to improve application performance. Software auto-tuning is a solution to this problem as it helps developers and system administrators to handle hundreds of system parameters. For example, current work in the literature use machine learning algorithms for Hadoop automatic configuration to improve performance. However, these solutions use single machine learning algorithms, thus making unfeasible to compare these solutions with each other to understand which approach is best suited given an application and its input. In addition, current work is intrusive or expose operational details for developers and/or system administrators. This work proposes a transparent, modular and hybrid approach to improve the performance of Hadoop applications. The approach proposes an architecture and implementation of transparent software that automatically configures the Hadoop. Furthermore, this approach proposes a hybrid solution that combines genetic algorithms with various machine learning techniques as separate modules. A research prototype was implemented and evaluated proving that the proposed approach can significantly reduce the execution time of applications Hadoop WordCount and Terasort autonomously. Furthermore, the approach converges quickly to the most suitable configuration application with low overhead.

Keywords: Self-configuration. MapReduce. Hadoop. Big Data. Cloud Computing. Genetic Algorithms. Machine Learning.

LISTA DE ILUSTRAÇÕES

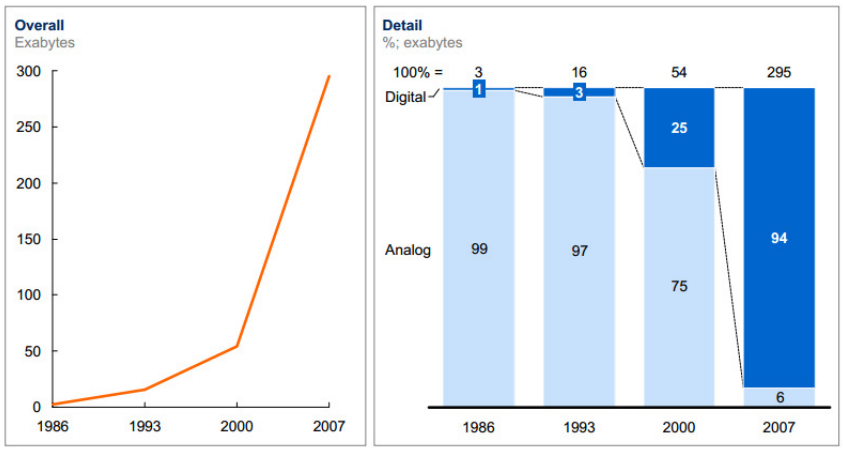
LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas, tem-se aumentado significativamente a quantidade de dados digitais gerados no mundo. Enquanto a quantidade de dados armazenados no mundo ultrapassou trezentos exabytes, os dados digitais passaram de um por cento a noventa e quatro por cento do total de dados (HILBERT; LÓPEZ, 2011), como ilustrado na Figura 1. Além disso, a Cisco prevê o crescimento de aproximadamente três vezes da quantidade de tráfego em redes Internet entre 2012 e 2017 (Cisco Systems, 2012), visto na Figura 2. Esse cenário torna-se ainda mais complexo devido à variedade dos tipos de dados que apresentam-se não apenas de modo uniforme, mas principalmente de maneira semi-estruturada ou não estruturada. Além disso, a grande velocidade na qual esses dados são gerados acompanha as tendências crescentes do volume e da diversidade de dados produzidos (ZIKOPOULOS; EATON; ZIKOPOULOS, 2011).

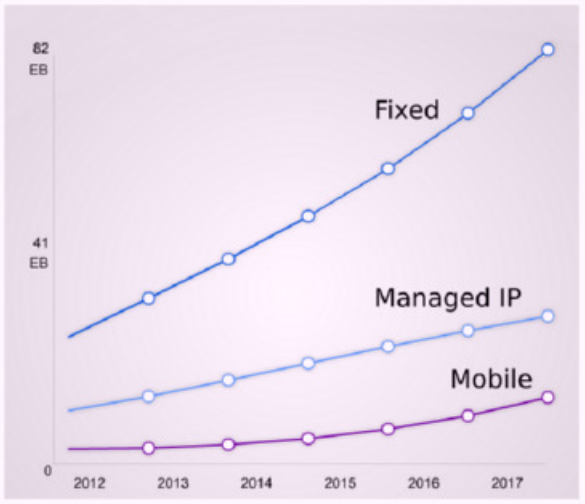
Figura 1 – Representação do aumento de dados de 1986 a 2007.



Fonte: McKinsey Global Institute (2011)

Dentre as tecnologias empregadas no processamento dessa grande quantidade de dados, destacam-se a Computação em Nuvem e o modelo de programação MapReduce. A Computação em Nuvem (VAQUERO et al., 2008) utiliza recursos computacionais virtuais sob demanda para processamento e armazenamento de dados intensivos (WANG et al., 2009). Por outro lado, o MapReduce (DEAN; GHEMAWAT, 2008) fornece um modelo de programação que facilita o desenvolvimento de aplicações que processam grande quantidade de dados. O MapReduce divide as etapas do processamento em funções *map* e *reduce*, deixando para os arcabouços MapReduce lidarem com detalhes de execução da aplicação. Desse modo, tem-se utilizado arcabouços MapReduce que executam suas tarefas *map* e *reduce* em infraestruturas de computação em nuvem para lidar com a crescente demanda de processamento de dados (BABU, 2010).

Figura 2 – Representação do crescimento do tráfego em redes IP.



Fonte: Cisco Systems (2012)

Embora o modelo de programação MapReduce facilite o desenvolvimento das aplicações que processam grandes quantidades de dados, o aumento da complexidade de configuração dos sistemas contemporâneos está aumentando os encargos sobre os desenvolvedores de aplicações. Por exemplo, o arcabouço MapReduce Hadoop (HADOOP, 2008) possui mais de duzentos parâmetros onde uma parte deles influencia diretamente no desempenho da execução das aplicações. O que torna essa situação ainda mais delicada é a natureza dinâmica dos sistemas distribuídos modernos, que podem apresentar mudanças no ambiente de execução. Assim, desenvolvedores e administradores de sistemas vêm dedicando cada vez mais tempo para a configuração ou ajuste de seus códigos e/ou dos sistemas de computação que os executam (TIWARI et al., 2009). A fim de ajudar com a complexidade de configuração de aplicações MapReduce que utilizam o Hadoop, muitas pesquisas nos últimos anos têm se dedicado à configuração automática de parâmetros do Hadoop. Essas pesquisas utilizam técnicas para avaliar um conjunto de mapeamentos de parâmetros alternativos e selecionar aquele que obtiver o melhor desempenho (TIWARI et al., 2009). Dessa maneira, a otimização de parâmetros de configuração tenta encontrar valores para um conjunto de parâmetros de tal forma que o desempenho seja maximizado (LIU et al., 2012). Desse modo, a configuração automática de parâmetros do Hadoop surge como uma possibilidade de facilitar a tarefa de desenvolvedores e administradores de sistemas, configurando-o para as necessidades específicas de cada aplicação.

Contudo, as abordagens encontradas na literatura para configuração automática do Hadoop utilizam soluções ineficientes ou específicas para o problema da configuração automática de parâmetros do Hadoop. Em outras palavras, essas abordagens realizam buscas exaustivas ou utilizam apenas algoritmos específicos de aprendizagem de máquinas, o que não permite a escolha adequada entre diferentes técnicas de aprendizado de máquina existentes. Além disso, essas soluções são intrusivas ao exporem detalhes operacionais para

programadores e/ou administradores de sistemas. Por exemplo, em Yigitbasi et al. (2013) e Liu et al. (2012), os autores utilizam a busca exaustiva para encontrar a configuração com melhor desempenho de aplicações do Hadoop. Buscas exaustivas são ineficientes por levarem muito tempo para encontrar a configuração adequada. Demais trabalhos empregam técnicas de aprendizagem de máquina para buscar configurações com melhor desempenho (LAMA; ZHOU, 2012), (WANG; LIN; TANG, 2012), (WU; GOKHALE, 2013), (BABU, 2010), (LIAO; DATTA; WILLKE, 2013), (LI et al., 2014) e (LI et al., 2014a). As técnicas de aprendizado de máquina utilizadas por esses modelos incluem *k-means*, *Support Vector Machine* (SVM), *Hill climbing*, *Simulated Annealing*, Algoritmos Genéticos. Por exemplo, na abordagem descrita em (LAMA; ZHOU, 2012) e (YIGITBASI et al., 2013) os autores treinam um modelo de SVM que é capaz de realizar uma previsão de maneira precisa e rápida do desempenho de uma aplicação para várias combinações de alocação de recursos, parâmetros de configuração e tamanhos de dados de entrada. Já os trabalhos que utilizam as técnicas *Hill climbing*((WANG; LIN; TANG, 2012) e (LI et al., 2014a)), *Simulated Annealing* ((WU; GOKHALE, 2013))e Algoritmos Genéticos ((LIAO; DATTA; WILLKE, 2013)) procuram a configuração ideal através de parâmetros a serem otimizados, simplificando o trabalho de procurar em um espaço dimensional alto através da busca de valores mínimos de tempo. Dentre esses trabalhos, destaca-se o Gunther (LIAO; DATTA; WILLKE, 2013), que utiliza Algoritmos Genéticos e consegue melhorar o desempenho das aplicações Hadoop. Por fim, a maior parte desses trabalhos apresenta soluções intrusivas que requerem modificação da aplicação e/ou do Hadoop.

Esse trabalho tem por objetivo propor uma abordagem transparente, modular e híbrida para melhorar o desempenho de aplicações Hadoop através da configuração automática de parâmetros. A abordagem propõe uma arquitetura e implementação de software não intrusivos que configura automaticamente o Hadoop sem modificá-lo e sem modificação no código da aplicação. Além disso, a abordagem inova ao propor uma solução híbrida que combina Algoritmos Genéticos e várias técnicas de aprendizado de máquina (*machine learning*) para a configuração automática estática para parâmetros do Hadoop que é transparente, independente, não intrusiva, modular e extensível. A arquitetura é modular, permitindo a utilização de técnicas específicas e a adição de demais técnicas em módulos separados. Um protótipo da pesquisa foi implementado e avaliado[1], mostrando que a abordagem proposta consegue diminuir significativamente o tempo de execução de aplicações Hadoop. O prototipo utiliza as seguintes técnicas de aprendizado de máquina: Redes de Funções de Base Radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP) e Perceptron de Múltiplas Camadas (MLP). Além disso, a abordagem consegue convergir rapidamente para uma configuração melhor para cada aplicação, alcançando baixos níveis de custos adicionais (*overhead*). Por último,

[1] Toda a pesquisa é acessível para uso, mais detalhes na seguinte página: http://gitlab.com/NathaliaUfal/ANICAH

o protótipo foi validado utilizando dois estudos de casos oficiais do Hadoop: WordCount e Terasort.

Esse trabalho é dividido em seis capítulos. Um capítulo diz respeito ao estado da arte deste estudo (Capítulo 2) que vai abranger tópicos como a Computação em Nuvem, MapReduce e algumas técnicas de Aprendizagem de Máquina. Após isso, o Capítulo 3 tem por objetivo o problema da configuração automática do Hadoop e mostrar os trabalhos existentes na literatura que focam esse mesmo problema, incluindo a contextualização e a motivação para o estudo. O Capítulo 4 mostra a contribuição desse estudo, expondo os objetivos pretendidos por esse trabalho, a arquitetura da abordagem proposta para configuração automática de parâmetros do Hadoop e a implementação do protótipo de pesquisa. O Capítulo de Avaliação (Capítulo 5) diz respeito à avaliação da abordagem através do protótipo implementado realizado para atestar e validar a eficiência e eficácia da solução. Por fim, o Capítulo 6 mostra as conclusões e os trabalhos futuros.

2 ESTADO DA ARTE

O Capítulo tem como função apresentar o levantamento do estado da arte acerca dos principais conceitos, paradigmas e tecnologias que foram utilizados nesse trabalho. Desse modo, esse Capítulo é dividido nas seguintes Seções: Computação em Nuvem (2.1) que definiu, caracterizou e expôs vantagens e desvantagens da computação em nuvem; MapReduce (2.2) que expôs características do modelo de programação MapReduce, o funcionamento desse arcabouço e a apresentação da ferramenta Hadoop.

2.1 Computação em nuvem

O aumento do poder computacional tem ocasionado a produção de um fluxo elevado de dados. Isto é, o progresso computacional, principalmente na tecnologia de Web, permite que qualquer usuário forneça e consuma facilmente muito conteúdo e em qualquer formato. Essa facilidade de produzir uma enorme quantidade de dados resultou na necessidade de uma mudança de paradigma na arquitetura de computação e mecanismos de processamento de dados em grande escala. A Computação em Nuvem (VAQUERO et al., 2008) está vinculada a esse novo paradigma para fornecer infraestruturas no intuito de reduzir os custos associados à gestão dos recursos de hardware e software (SAKR et al., 2011).

De maneira resumida, o termo "Nuvem" refere-se à oferta da computação como utilidade de maneira transparente. Ou seja, a oferta de hardware, de software, o acesso a dados e serviços de armazenamento que não irão exigir o conhecimento do usuário final da localização física e configuração da arquitetura que está entregando os serviços (JADEJA; MODI, 2012).

Segundo o relatório técnico de Mell e Grance (2011), a Computação em Nuvem é um modelo que permite o acesso à rede ubíqua sob demanda a recursos computacionais configuráveis compartilhados (como redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com um esforço mínimo de gerenciamento ou interação com o provedor de serviços. Tal modelo de nuvem é composto de cinco características essenciais.

Primeira característica O consumidor pode usufruir das capacidades de computação sob demanda automaticamente sem a necessidade de interação humana com o provedor de serviço.

Segunda característica Os recursos estão disponíveis através da rede e são acessados por meio de mecanismos que promovem o uso de plataformas heterogêneas de cliente (celulares, tablets, notebooks e estações de trabalho).

Terceira característica Os recursos de computação do provedor são reunidos para servir vários consumidores usando um modelo *multi-tenant*, com diferentes recursos físicos e virtuais atribuídos e realocados dinamicamente de acordo com a demanda do consumidor. Nesse caso, há um senso de independência local em que o cliente geralmente não tem controle ou conhecimento sobre a localização exata dos recursos disponibilizados.

Quarta característica Os recursos podem ser elasticamente provisionadas e liberadas, em alguns casos, automaticamente, para escalar rapidamente para fora e para dentro, compatível com a demanda. Para o consumidor, os recursos disponíveis para a provisão de muitas vezes parece ser ilimitado e pode ser apropriado em qualquer quantidade a qualquer momento.

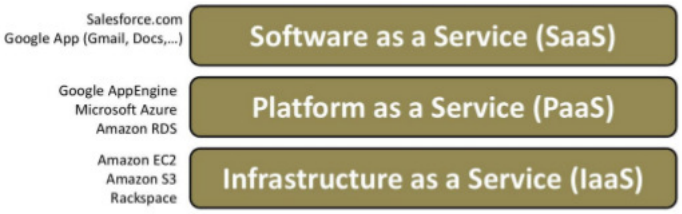
Quinta característica Sistemas em nuvem controlam e otimizam automaticamente o uso dos recursos, aproveitando a capacidade de medição em algum nível de abstração adequado para o tipo de serviço. O uso de recursos pode ser monitorado, controlado e reportado, oferecendo transparência tanto para o fornecedor como para o consumidor do serviço utilizado.

Um ponto importante dentro do conceito de Nuvem é o modelo *pay-as-you-go*. De acordo com (LI et al., 2014b), o mecanismo de preços mais comum usado pela nuvem é o modelo *pay-as-you-go*, onde os usuários pagam um preço fixo por unidade de recursos por unidade de tempo. Desse modo, segundo (TANG; LEE; HE, 2014), é necessário garantir que a quantidade de recursos obtidos por cada usuário é proporcional ao seu pagamento. Isto é, nessa modalidade paga-se um valor periódico, como se fosse uma assinatura, pagando pelos recursos utilizados e pelo tempo que se utilizou.

Além disso, a computação em nuvem é composta de três modelos de serviço, como mostra a Figura 3. O primeiro é a Infraestrutura como Serviço (IaaS) onde se localiza o fornecimento de recursos como servidores, largura de banda de rede, armazenamento e ferramentas necessárias para construir um ambiente de aplicação a partir do zero. A segunda camada é a Plataforma como Serviço (PaaS) que fornece um ambiente de alto nível onde os desenvolvedores podem escrever aplicações personalizadas. Finalmente, a camada de Software como Serviço (SaaS) refere-se ao software de propósito específico disponibilizado através da Internet. Ele não exige que cada usuário final baixe manualmente, instale, configure, execute ou utilize os aplicativos de software em seus próprios ambientes de computação.

Ademais, o serviço de nuvem é classificado como privado, público ou híbrido. Na nuvem privada a infraestrutura de nuvem trabalha inteiramente para uma organização e pode estar localizada interna ou externamente. Na nuvem pública a infraestrutura de nuvem está disponível para o público em geral ou um grupo específico e é de propriedade

Figura 3 – Modelos de serviço da Computação em Nuvem.



Fonte: SAKR et al. (2011)

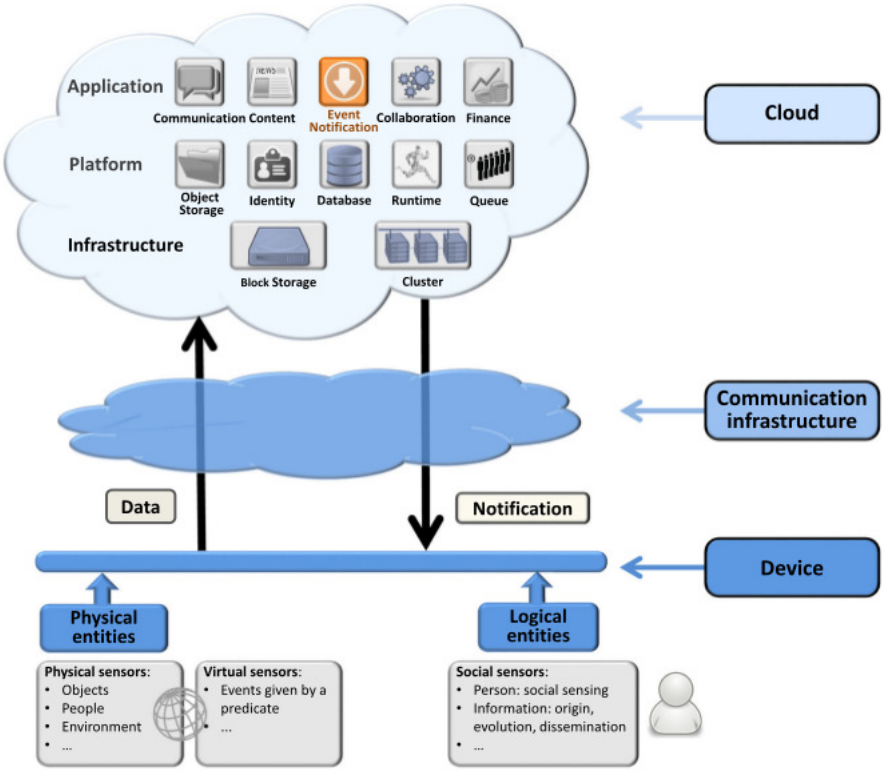
de uma organização que negocia serviços de nuvem. Por fim, na nuvem híbrida a infraestrutura de nuvem é uma composição de mais de um modelo de nuvem que funcionam como entidades separadas, mas usam tecnologias padrões que possibilitam a portabilidade de dados ou aplicações.

Segundo Boukerche et al. (2014), atualmente é comum usuários armazenarem e terem acesso a informações na nuvem usando dispositivos móveis. Dessa maneira, os usuários móveis esperam acessar continuamente tais recursos onde e quando eles quiserem. Esta tecnologia acaba com a noção de *placeness*. Há pouco tempo, os recursos computacionais exigiam que os usuários estivessem, obrigatoriamente, em um determinado lugar físico para ter acesso a informações. Com a Computação em Nuvem e Computação Móvel, os usuários exigem que a informação relevante seja acessada em qualquer contexto físico e lógico. A Figura 4 mostra o dispositivo, que é usado para recolher dados a partir de diferentes fontes heterogêneas; a infraestrutura de comunicação, responsável por toda a comunicação entre a nuvem e os usuários; e a nuvem, que é responsável por todo o processamento. Isto apoia a ideia de que *placeness* e contexto são dois pilares para fornecer informações personalizadas a um determinado usuário móvel.

Para conferir qualidade aos serviços complexos na nuvem (*Quality of Service* - QoS), são firmados com os clientes os Acordos em Nível de Serviço (*Service-Level Agreements* - SLAs). Uma grande preocupação por parte dos provedores de serviços em nuvem é estabelecer os termos do SLA acordados, no intuito de evitar perdas e penalidades. Desse modo, é necessário traduzir a qualidade de serviço para configurações em baixo nível, para que sejam capazes de garantir o cumprimento dos termos acordados. Entretanto, os sistemas atuais não têm fornecido suporte integrado para especificação, tradução e aplicação dos SLAs. Alguns trabalhos na literatura vêm dedicando esforços para especificar e aplicar SLAs para provedores de serviços em nuvem como no estudo de (FREITAS; PARLAVANTZAS; PAZAT, 2012).

A grande vantagem da Computação em Nuvem é o fato de utilizar aplicações que não estão instaladas no computador do usuário, acessando-as através da Internet. Isto é, não há preocupações com detalhes como hardware do computador pessoal utilizando os dados da nuvem independente desses fatores. Outra vantagem é que as atualizações

Figura 4 – Computação em Nuvem para a Computação Móvel.



Fonte: BOUKERCHE et al. (2014)

colaborativas e o compartilhamento de arquivos são simplificados. Isso acontece porque os dados manipulados estão sempre no mesmo "lugar". Por fim, o usuário não precisa se preocupar com a manutenção da infraestrutura física ou de instalação de softwares. Tudo isso é responsabilidade do provedor da nuvem.

Uma das principais desvantagens do uso da Computação em Nuvem é que se for requisitada uma alta taxa de transferência e a internet não tiver boa largura de banda, o acesso pode ser prejudicado. Outra desvantagem diz respeito a questões como segurança e privacidade das informações armazenadas.

Por fim, o Hadoop como implementação do modelo de programação MapReduce (que será mostrado em 2.2) é uma das mais populares tecnologias para processamento de grandes quantidades de dados nos ambientes de nuvem pública ou privada. Então, a Computação em Nuvem vai fornecer a estrutura física fundamental para o processamento paralelo do Hadoop.

2.2 MapReduce

Nas últimas décadas, a quantidade de dados gerados tem aumentado significativamente. Com isso, muitos fatores influenciaram no aumento da produção de dados, como o surgimento da computação em nuvem permitindo o acesso a diversas máquinas virtuais,

armazenamento e recursos relacionados a redes. Nesse contexto, o modelo de programação MapReduce propõe fornecer uma grande abstração para o processamento dos dados intensivos com o auxílio da nuvem. Desse modo, essa Seção apresenta o funcionamento do modelo de programação MapReduce na Subseção 2.2.1, os detalhes do funcionamento do Arcabouço MapReduce na Subseção 2.2.2 e apresenta o Hadoop que é a distribuição mais conhecida do MapReduce na Subseção 2.2.3.

2.2.1 Modelo de programação

Para muitas aplicações (como processamento de logs, *crawling* de páginas da web, ordenações, criação de índices para buscas) os cálculos relativos a elas são conceitualmente simples. Entretanto, os dados de entrada são geralmente grandes e os cálculos têm que ser distribuídos em centenas ou milhares de máquinas, a fim de terminar em uma quantidade razoável de tempo (DEAN; GHEMAWAT, 2008). No entanto, o desenvolvimento de software voltado para ambientes distribuídos é muito complexo. Desse modo, a Google propôs o modelo de programação MapReduce com o intuito de permitir expressar os cálculos simples para grandes quantidades de dados e facilitar tais atividades para o programador, para que o programador possa focar a atenção nos detalhes da aplicação.

O modelo de programação MapReduce funciona da seguinte maneira. A execução toma como entrada um conjunto de chave/valor e produz um conjunto de saída chave/valor. Essa computação é expressa nas funções *map* e *reduce*. A função de mapear recebe um par de entrada e produz um conjunto de pares de chave/valor intermediário. O MapReduce junta todos os valores intermediários associados a uma mesma chave intermediária I e as passa para a função de reduzir. A função de reduzir aceita uma chave intermediária I e um conjunto de valores atribuídos a essa chave. Por fim, esses valores são fundidos para formar um conjunto, possivelmente menor.

Por exemplo a aplicação Wordcount que é uma aplicação que tem o objetivo de contar as ocorrências das palavras em um conjunto de arquivos de textos. A implementação das funções *map* e *reduce* são mostradas no código da Figura 5. Na função *map*, a chave é o nome do arquivo de texto e o valor é o conteúdo desse arquivo. Para cada palavra existente no conteúdo do arquivo, é emitida uma chave intermediária, que é a palavra, e o valor é 1 (relativo a uma ocorrência da palavra). Após isso, todas as ocorrências de uma mesma palavra são agrupadas e passadas para função *reduce*. Na função *reduce*, a chave é uma palavra e o valor é um conjunto de contadores associados a essa palavra. As ocorrências nesse conjunto de contadores são somadas em uma variável que gerará um par chave/valor que é a palavra e o número geral de ocorrências.

O modelo de programação MapReduce é inspirado nas primitivas de mapear e reduzir presentes em Lisp e muitas outras linguagens funcionais. Esse trabalho seminal de MapReduce, compartilha a experiência da aplicação do modelo e as melhorias que ocorreram desde o seu primeiro lançamento em fevereiro de 2003.

Figura 5 – Pseudocódigo da aplicação Wordcount.

```
map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
      EmitIntermediate(w, "1");

reduce(String key, Iterator values):
   // key: a word
   // values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);
   Emit(AsString(result));
```

Fonte: DEAN; GHEMAWAT (2008)

Entretanto, duas questões ainda não ficaram claras. Em primeiro lugar, como garantir a qualidade das funções de mapear e reduzir para um problema. Isto é, existem muitas maneiras de implementar a solução para um mesmo problema. Entretanto, é difícil medir ou avaliar se aquela implementação resolve o problema da melhor maneira. Segundo, que não se encontra claro como decidir corretamente a granularidade das tarefas, embora existam fórmulas, porém é um processo intuitivo. Ou seja, é difícil determinar quais tarefas devem pertencer a função *map* ou para a função *reduce*, e o que deve ser envolvido no processamento MapReduce.

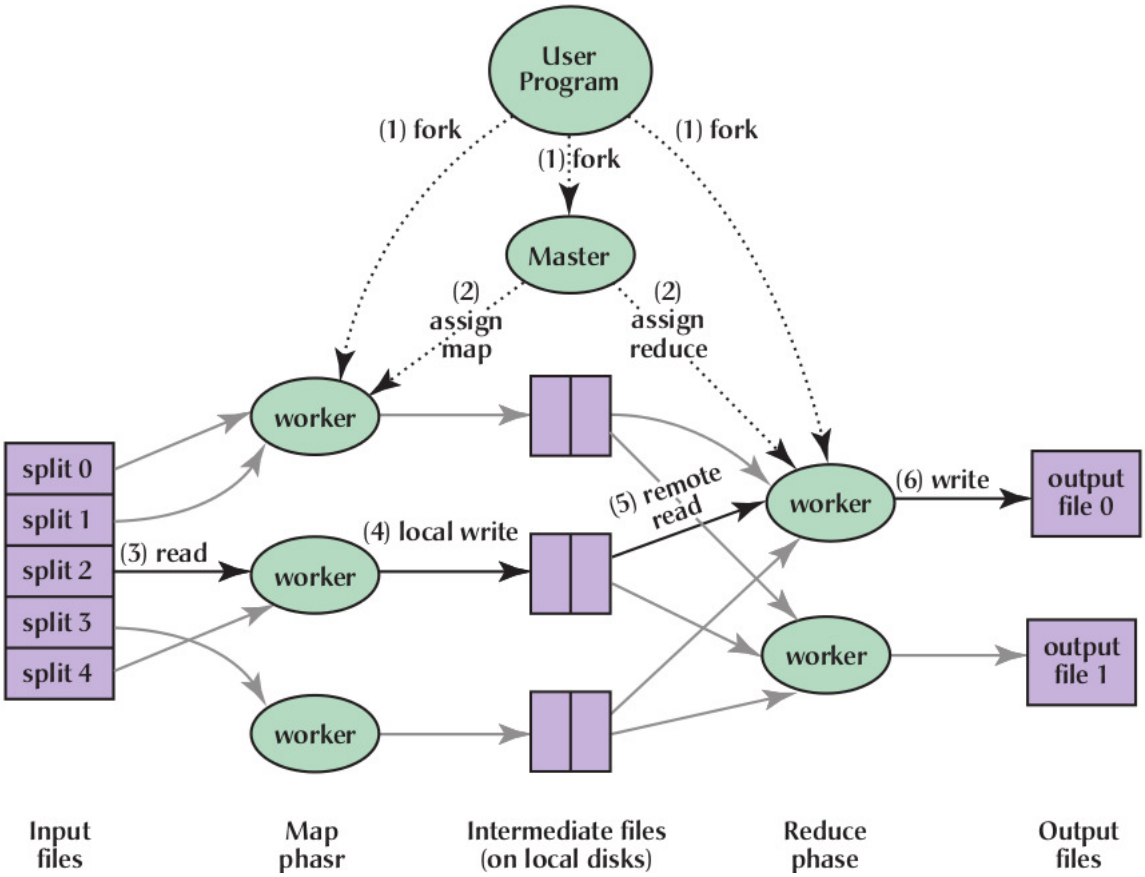
2.2.2 O arcabouço MapReduce

O arcabouço (*framework*) MapReduce permite que aplicações possam ser executadas sem expor detalhes operacionais aos desenvolvedores. Ele se baseia em um sistema de arquivos distribuído onde os dados das aplicações MapReduce são armazenados. O fluxo geral da execução de uma aplicação no arcabouço MapReduce é ilustrado pela Figura 6. Existem várias cópias do programa no cluster. Uma delas é chamada de nó mestre cuja função é requisitar os trabalhadores ociosos e designar para cada um uma tarefa *map* ou *reduce*. As demais cópias são chamadas de nós trabalhadores aos quais são atribuídos trabalhos pelo mestre. O nó mestre chama internamente a função MapReduce, pois a aplicação estará armazenada nele e ele designará as outras atividades para os nós escravos. Primeiramente, o arcabouço MapReduce divide os arquivos de entrada em M pedaços (*splits*) normalmente de tamanho 16-64MB[2]. Em seguida, o arcabouço inicia cópias do programa (*forks*) em um cluster de máquinas previamente configurados.

Existem dois tipos de tarefas a serem atribuídas aos trabalhadores: função *map* e função *reduce*. Os nós trabalhadores que executam as tarefas *map* leem o conteúdo das

[2] Esse valor é configurável através de um parâmetro opcional.

Figura 6 – Fluxo de execução do arcabouço MapReduce.



Fonte: DEAN; GHEMAWAT (2008)

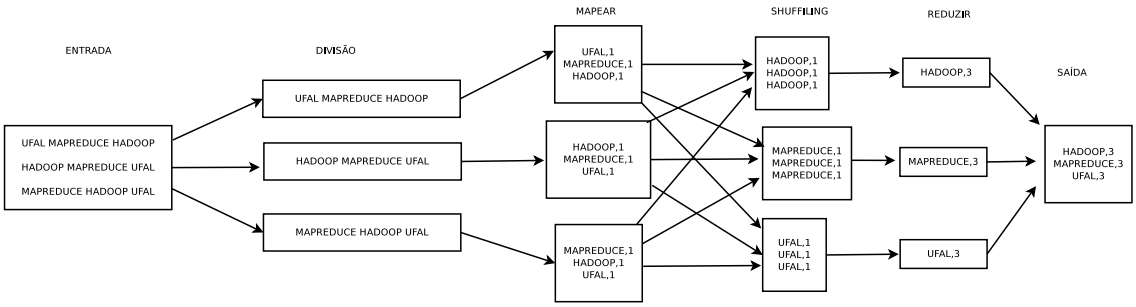
entradas correspondentes (*splits*). Eles analisam pares de chave/valor de entrada. Os pares intermediários produzidos pela função *map* ficam no *buffer* de memória. Periodicamente, os pares presente no buffer são gravados em disco local, divididos em R regiões pela função de particionamento. Os locais destes pares de *buffer* no disco local são passadas de volta para o mestre que é responsável por encaminhar esses locais para os trabalhadores responsáveis pela função *reduce*.

Um trabalhador *reduce* é notificado pelo mestre sobre estes locais para ler tais dados do *buffer* a partir dos discos locais dos trabalhadores *map*. Quando um trabalhador *reduce* leu todos os dados intermediários para a sua partição, ele os classifica pelas chaves intermediárias para que todas as ocorrências de uma mesma chave sejam agrupadas. O trabalhador *reduce* repete a chamada da função *reduce* ao longo dos dados intermediários classificados e para cada chave única intermediária encontrada, o nó mestre passa a chave e o respectivo conjunto de valores intermediários para a função *reduce*. A saída da função *reduce* é anexada a um arquivo de saída final. Quando todas as tarefas *map* e *reduce* foram concluídas, o mestre acorda o programa do usuário, que continua o seu fluxo de execução.

A Figura 7 mostra um exemplo do funcionamento do arcabouço MapReduce para a

aplicação Wordcount. Seja um arquivo de entrada com três linhas. Cada uma dessas linhas é atribuída a um *mapper* que realizará a função de Mapear: cada palavra da linha que é atribuída a ele gerará um par de chave/valor, onde a chave é cada palavra e o valor é 1. Ao final da função de Mapear, será obtido um conjunto de pares chave e valor intermediários para aquela linha. Depois disso, a etapa de *shuffling* vai organizar os pares chaves/valor gerados pelos *mappers*, unindo os pares com a mesma chave. Após isso, esses pares intermediários são atribuídos aos *reducers* que irão fazer a função de Reduzir: somar as ocorrências para a mesma chave e gerar um conjunto possivelmente menor de pares.

Figura 7 – Etapas do funcionamento da aplicação Wordcount no arcabouço MapReduce.



Fonte: Elaborada pelo autor. (2015)

O MapReduce ganhou popularidade não somente devido à sua interface expressiva, mas também devido ao seu suporte à tolerância a falhas e desempenho. Mais detalhes sobres esses aspectos são discutidos a seguir.

Desempenho Em relação ao desempenho, a largura de banda de rede é um recurso relativamente escasso no ambiente de computação distribuída. Para economizar largura de banda de rede, os dados de entrada são armazenados localmente e são divididos em blocos de 64MB. Além disso, várias cópias de cada bloco (normalmente 3 cópias) são armazenadas em diferentes máquinas. O mestre MapReduce tem as informações de localização dos arquivos de entrada e tenta agendar uma tarefa *map* em uma máquina que contenha uma réplica correspondente ao dado de entrada. Se falhar, ele tenta agendar uma tarefa *map* mais perto de uma réplica dos dados de entrada. Desse modo, a maioria dos dados de entrada é lida localmente e não consome largura de banda de rede.

Tolerância a falhas O arcabouço MapReduce possui dois principais mecanismos de tolerância a falhas. No primeiro mecanismo, o nó mestre do arcabouço MapReduce chama cada nó trabalhador periodicamente. Caso nenhuma resposta seja recebida por um trabalhador num certo intervalo de tempo, o mestre marca o trabalhador como falho. O segundo mecanismo de tolerância a falhas repete execuções dos nós trabalhadores. Toda tarefa *map* é executada novamente em caso de falhas para que os dados intermediários fiquem acessíveis. Isso acontece porque a saída da função *map* é armazenada no sistema de arquivo local da máquina do trabalhador em que foi executado. Já as tarefas *reduce*

que falharam não precisam ser reexecutadas uma vez que a sua produção é armazenada no sistema de arquivos distribuídos. Comportamentos como esse permitem que o MapReduce seja resistente a falhas nos nós trabalhadores.

Além dos dois mecanismos de tolerância a falhas, o arcabouço MapReduce também lida com nós trabalhadores retardatários. Uma das causas mais comuns de aumento do tempo total necessário para uma operação MapReduce é o *straggler*. O *straggler* é quando um nó trabalhador leva um longo tempo para concluir algumas das últimas tarefas *map* ou *reduce* na computação. O arcabouço MapReduce possui um mecanismo para atenuar o problema dos nós trabalhadores retardatários. Quando uma operação MapReduce está próxima da conclusão, o mestre realiza execuções de backup das tarefas restantes em andamento. A tarefa é marcada como concluída sempre que o principal ou a execução de backup for concluída. Apesar desse mecanismo aumentar os recursos computacionais utilizados na operação, ele permite melhorar a tolerância a falhas do arcabouço.

2.2.3 Hadoop

Visão geral

Tendo acontecido a ascensão do MapReduce, surgiu o Hadoop (HADOOP, 2008) que, de um modo geral, foi quem mostrou o MapReduce como solução para o processamento paralelo de dados e concedeu a ele a visibilidade que se tem nos dias atuais. O Hadoop é um projeto que oferece uma solução para problemas relacionados a *Big Data*, ajudando a construir aplicações robustas envolvendo dados massivos e importantes para as organizações que os manipulam.

Atualmente, a computação em nuvem e implementações de código aberto Hadoop têm ganhado muita pesquisa e diferentes abordagens têm sido desenvolvidas para otimizar as configurações de maneira automática e eficiente. Desse modo, o Hadoop tem sido associado frequentemente com a computação em nuvem e conjuntos de dados em larga escala. Um grande número de empresas, tais como Yahoo!, Facebook, Twitter, NetFlix, Amazon e outras, usam o Hadoop para aplicações tais como a indexação Web, mineração de dados, geração de relatórios, análise de log, análise financeira, simulação científica e investigação de bioinformática.

O Hadoop é uma coleção de subprojetos relacionados para computação distribuída, sendo todos eles hospedados pela *Apache Software Foundation*. O Apache Hadoop é um arcabouço (*framework*) de código aberto que implementa o modelo de programação MapReduce que dá suporte a escalabilidade e confiabilidade para a computação de grandes quantidades de dados (WU; GOKHALE, 2013). Além disso, o Hadoop localiza-se na camada Plataforma como um Serviço (PaaS) da Computação em Nuvem, isto é, o Hadoop ajuda a fornecer um ambiente para que desenvolvedores que lidam com grandes dados criem mais facilmente suas aplicações. Esse arcabouço é composto de quatro módulos:

Hadoop Common contém as utilidades comuns que suportam os outros módulos do Hadoop.

HDFS é sistema de arquivos distribuído e confiável, responsável pelo armazenamento dos dados.

Hadoop YARN é um *framework* para a programação de trabalho e gestão de recursos de cluster.

Hadoop MapReduce é um sistema baseado em YARN para o processamento paralelo de grandes conjuntos de dados. No Hadoop, o nó mestre é chamado *JobTracker* e os nós escravos são chamados *TaskTracker*.

As desvantagens são: existência de um único nó mestre; se utilizado para arquivos pequenos podem adquirir efeito contrário, isto é, pode levar mais tempo do que levaria sem a ajuda do Hadoop; não pode ser utilizado para problemas não paralelizáveis; não pode ser aplicado a problemas cujas tarefas são dependentes entre si. Por outro lado, as vantagens de usar o Hadoop residem na questão de possuir o código aberto, ser econômico (por ser código aberto, além de utilizar máquinas e redes convencionais), robusto, escalável, personalizável e permite que o desenvolvedor foque no problema.

Portanto, o Hadoop facilitou e barateou a análise e o acesso a grandes dados. Para isso, os engenheiros não precisam mais lidar com problemas de infraestrutura de execução no tratamento de dados, ou seja, não precisam lidar com problemas de escalabilidade, tolerância a falhas, desempenho, dentre outros desafios.

Ciclo de vida de uma aplicação Hadoop

O ciclo de vida padrão de uma aplicação é mostrado na Figura 8. Inicia-se com o desenvolvedor programando uma aplicação MapReduce através da utilização de bibliotecas Hadoop. Nesse momento, ele irá modelar o problema por meio da especificação de como funcionarão as funções Map e Reduce para determinado domínio. Essa etapa é fundamental para a qualidade da saída que será produzida.

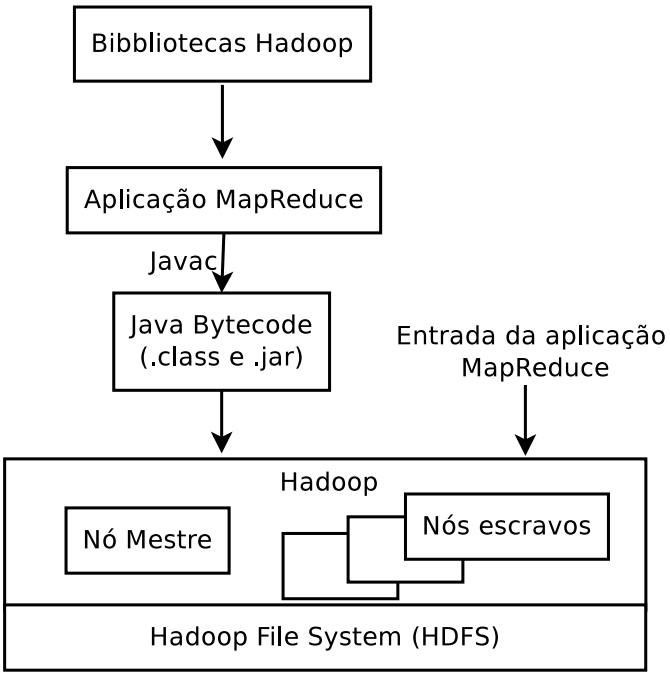
Tendo sido implementada a aplicação Hadoop, ela será compilada pelo Javac que lê definições de classe e de interface, escritas na linguagem de programação Java, e compila em arquivos de classe Java bytecode. Ele também pode processar anotações em arquivos fonte e classes Java[3]. Desse modo, o Java bytecode gerado será invocado para ser executado no Hadoop.

Tal aplicação possui dados de entrada que precisam ser armazenados no sistema de arquivos distribuído do Hadoop (HDFS) para poder ser acessado durante a execução. Tendo isso, invoca-se um comando no terminal especificando a localização da aplicação

[3] http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javac.html

criada (.jar), a classe principal para a chamada das funções MapReduce (.class), a localização da entrada e da saída no HDFS. Invocando-se o comando no formato padrão a aplicação é executada com a parametrização *default* do Hadoop produzindo uma saída que é armazenada no HDFS, podendo facilmente ser copiada como arquivo na máquina local.

Figura 8 – Ciclo de vida de uma aplicação Hadoop.



Fonte: Elaborada pelo autor. (2015)

Configuração paramétrica do Hadoop

O Hadoop possui mais de 200 parâmetros configuráveis, mostrados no Apêndice **??**, fornecendo aos usuários a flexibilidade para personalizá-lo de acordo com sua necessidade (YIGITBASI et al., 2013). Assim, o Hadoop é parametrizado para permitir aos usuários gerenciar o fluxo de dados nas fases de *map*, *reduce* e *shuffle*. Alguns desses parâmetros têm impacto significativo no desempenho. Por exemplo, existe uma interação entre parâmetros p e p' quando a magnitude do impacto que a variação p tem sobre o desempenho do trabalho depende da configuração específica de p'. Além disso, as características de infraestrutura também podem impactar consideravelmente no desempenho.

Em (LI et al., 2014), os autores dividem os parâmetros do Hadoop em três tipos onde:

Categoria A A maioria dos parâmetros é configurada para a operação padrão e não tem impacto no desempenho. Por exemplo, *dfs.datanode.address* que configura informações da porta do *namenode*.

Categoria B Alguns parâmetros podem ser definidos quando o trabalho é submetido. Para executar o programa em Hadoop, o sistema irá criar um objeto de configuração de trabalho com base em alguns parâmetros dados dos usuários. Este objeto de configuração de trabalho geralmente é altamente relevante para o desempenho.

Categoria C Para além dos parâmetros de configuração de trabalho, cujos valores são especificados explicitamente pelos utilizadores, existe um grande número de parâmetros cujos valores são especificados implicitamente a partir do sistema.

Além disso, todos estes parâmetros (com exceção da Categoria A) controlam diversos aspectos do trabalho durante a execução, tais como alocação de memória, otimização de entrada e saída e uso de banda de rede.

Há também na literatura outra classificação de parâmetros do Hadoop, como proposto em (WU; GOKHALE, 2013):

Parâmetros fundamentais são usados para definir as características mais importantes de um cluster MapReduce. Como por exemplo o tamanho do *buffer*.

Parâmetros relevantes para a MapReduce são relevantes para o procedimento MapReduce, alguns deles tem um efeito direto apenas na fase de mapear ou reduzir, enquanto outros podem ter efeito em ambas as fases. Como por exemplo, a especificação do número de *reducers*.

Parâmetros relevantes para o HDFS os parâmetros, tais como a especificação de quantas réplicas devem ser armazenadas, que pertencem ao grupo HDFS.

Já de acordo com (LI et al., 2014a), os parâmetros do Hadoop também podem ser classificados em três categorias. Isso é feito com base em *quando* o valor modificado de um parâmetro pode ter efeito. Os autores salientam que os valores ótimos de cada um dos parâmetros dependem das características das aplicações, o tamanho e o conteúdo dos dados de entrada associados e o *setup* do cluster. Assim, um valor ótimo é obtido quando o desempenho é maximizado. As categorias de parâmetros propostas por (LI et al., 2014a) são descritas a seguir.

Grupo 1 Inclue parâmetros que são difíceis de serem modificados depois que a aplicação foi iniciada. Exemplos: número de *mappers*, número de *reducers* e valor de *slow start*.

Grupo 2 Consiste dos parâmetros que não podem ser alterados em tempo real para as tarefas já em execução, mas podem impactar as tarefas que serão lançados a partir de mudanças que foram feitas. Por exemplo, *io.sort.mb* que corresponde à quantidade total de *buffer* de memória para usar durante a ordenação de arquivos, em megabytes. Escolhendo os valores corretos para esta categoria pode haver uma redução de Entrada/Saída e melhorar a utilização do cluster.

Grupo 3 É composto de parâmetros que podem ser alterados em tempo real e entram em vigor imediatamente. Exemplos: *mapred.inmemmerge.threshold* (corresponde ao limite, em termos do número de arquivos para o processo de *merge* na memória) e *io.sort.spill.percent* (relativo ao limite suave no *buffer* ou nos *buffers* de coleção de registros).

2.3 Abordagens para configuração automática parametrizada

De acordo com Kephart e Chess (2003), um manifesto lançado pela IBM detectou que um grande obstáculo que impedia a marcha do progresso na computação é a complexidade do software que caminhava para uma crise iminente. Os sistemas atuais se mostram cada vez mais interconectados e heterogêneos. Desse modo, torna-se muito difícil prever o funcionamento do sistema, fazendo com que questões importantes tenham que ser tratadas em tempo de execução. Isso atribui uma grande complexidade às aplicações, como MapReduce, para encontrar soluções efetivas em um tempo hábil diante das mudanças contínuas e necessidades conflitantes.

Uma das maneiras de abordar tal crise é a Computação Autonômica (HORN, 2001) onde os sistemas de computação podem gerenciar a si mesmos para atingir objetivos de alto nível. Em geral, modelos de desempenho são usados para ajustar de maneira automática essas diversas aplicações complexas. Uma das abordagens mais comuns envolve modelos de aprendizado de máquina. Esses modelos têm sido propostos em muitos campos para estimar o desempenho de sistemas complexos, exigindo grandes conjuntos de treinamento, a fim de construir um modelo satisfatório (LIAO; DATTA; WILLKE, 2013).

As Seções a seguir apresentam as abordagens de otimização e aprendizagem de máquina mais frequentemente empregadas para configuração automática de aplicações tanto para encontrar mínimos ou máximos globais como Algoritmos Genéticos (2.3.1) e modelos para predição de desempenho de sistemas como Regressão Linear (2.3.2), Redes de Funções de Base Radial (2.3.4), Máquinas de Vetores de Suporte (2.3.5), Processos Gaussianos (2.3.6) e Perceptron de Múltiplas Camadas (2.3.3).

2.3.1 Algoritmos Genéticos (AG)

A área de otimização estuda os métodos capazes de encontrar a solução ótima para um problema de melhoramento (POSK; HUYER; PAL, 2012). Sabendo que as simulações são realizadas para modelos com aspectos dinâmicos, o trabalho de (MAGOULAS; ELDABI; PAUL, 2002) mostra três categorias para otimizações de variáveis quantitativas.

- Utiliza o método de tentativa e erro, variando as variáveis de entrada no intuito de encontrar o conjunto que obtenha os melhores resultados.

- Varia sistematicamente as variáveis de entrada, para observar seus efeitos nas variáveis de saída.

- Utilização de uma abordagem automatizada para a otimização.

Nas estratégias de busca global para automatização da otimização se encontram os métodos como Recozimento Simulado (*Simulated Annealing*) (CORANA et al., 1987), Inteligência de Enxame (*swarm intelligence*) (EBERHART; SIMPSON; DOBBINS, 1996) e Algoritmos Genéticos (*Genetic algorithms*) (MICHALEWICZ, 1994). Nesse trabalho, as otimizações de desempenho das aplicações MapReduce pretendidas utilizaram Algoritmos Genéticos para buscas nesse contexto.

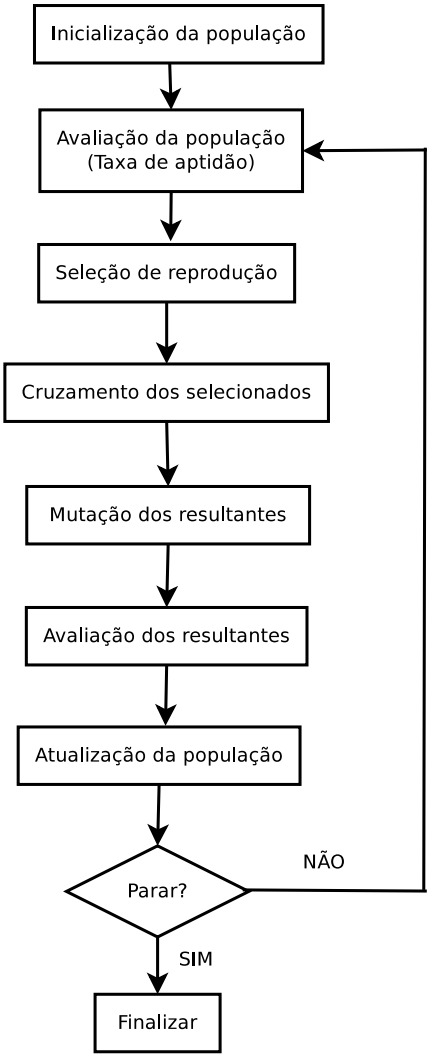
Em suma, os Algoritmos Genéticos (AG) são algoritmos inspirados na teoria da evolução, seleção natural e nos processos biológicos envolvidos. As soluções são representadas através de uma sequência de elementos (como o DNA), chamadas de indivíduos. Através de um critério de seleção, os indivíduos são avaliados como bons ou ruins e é decidido quais indivíduos passam para a próxima fase. Depois disso, os indivíduos devem ser cruzados, isto é, será usado parte de duas das soluções obtidas na etapa anterior que serão mescladas para gerar uma nova solução. Então, acontece a mutação que permite inserir características novas nas populações existentes, sendo elas melhores ou piores para solucionar o problema. Por fim, esses passos devem ser aplicados diversas vezes gerando várias soluções.

Alguns dos principais componentes de um Algoritmo Genético são descritos a seguir. O indivíduo é a unidade básica do AG que é responsável por codificar as soluções e obter as respostas através de sua manipulação. Nesse panorama, dois conceitos são importantes: genótipo (informação presente na estrutura dos dados) e fenótipo (resultado da decodificação de um genoma). As características desejáveis são propagadas a gerações subsequentes através do cruzamento de dois ou mais indivíduos, algumas delas são modificadas de maneira marginal através das mutações. O AG vai determinar e manipular a frequência que determinadas sequências de genes aparecem na população.

O funcionamento detalhado dos Algoritmos Genéticos, mostrados na Figura 9, são descritos a seguir. A inicialização de um Algoritmo Genético é a criação de uma população inicial sobre a qual serão realizadas os próximos passos do algoritmo. Em geral, a geração é realizada de maneira aleatória para gerar os indivíduos com a maior diversidade possível. Depois disso, a avaliação torna-se responsável por cada indivíduo da população para que seja determinado quão boa é a resposta para o problema. A seleção é responsável por perpetuar características da espécie que sejam positivas. Nesse estágio os indivíduos são escolhidos para posterior cruzamento, através do uso da função de aptidão (ou *fitness*).

Na reprodução, uma vez que os indivíduos foram selecionados, vão passar pelo processo de cruzamento (*crossover*), onde partes desses genes são combinadas. Os operadores de cruzamento combinam as informações genéticas de dois indivíduos para gerar novos

Figura 9 – Representação das etapas da execução de um Algoritmo Genético de acordo com um critério de parada.



Fonte: ZINI (2009)

indivíduos (podendo ser em um ponto, em múltiplos pontos ou uniforme). A mutação vai operar sobre os indivíduos do processo de cruzamento alterando algo na sua estrutura. Na atualização, os indivíduos do cruzamento e da mutação são inseridos na nova população. A finalização consiste na verificação de uma condição que põe fim ao processo de evolução como número de gerações ou grau de convergência.

Por fim, um AG simples é mostrado no Algoritmo 1. Com a população inicializada, os indivíduos se reproduzem, cruzam, passam por mutações e avaliações que se repetem

até que um critério de parada seja alcançado.

Algoritmo 1: ALGORITMO GENÉTICO SIMPLES

```
1  Seja P(t) uma população da geração t
2  t ← 0
3  Inicializa P(t)
4  Avalia P(t)
5  enquanto o critério de parada não for satisfeito faça
6  |  t ← t + 1
7  |  Seleciona-se P(t) a partir de P(t-1)
8  |  Aplicar-se crossover sobre P(t)
9  |  Avaliar P(t)
10 fim
```

O Algoritmo Genético surge no intuito de encontrar possíveis soluções para um problema que são avaliadas com uma função de aptidão, que é construída de forma personalizada para o problema. Aqueles que tiverem os melhores valores de aptidão são considerados melhores soluções, tendo em vista que dependendo do domínio do problema a maneira como os os melhores são escolhidos variam. A principal desvantagem do Algoritmo Genético é complexidade para a modelagem de um AG que consiga abstrair o contexto e resolver a problemática em questão.

Os Algoritmos Genéticos apresentam-se como uma técnica satisfatória para busca global comparada com técnicas como *Simulated Annealing* (SA), *Hill Climbing*(HC), *Particle Swarm Optimization* (PSO) e *Recursive Random Search*(RRS) (LIAO; DATTA; WILLKE, 2013). De um modo geral, essas técnicas de busca global, como aos Algoritmos Genéticos, podem ser usadas para os casos onde há um grande número de possibilidades (como configurações possíveis para parâmetros do Hadoop) e precisa-se encontrar a melhor solução em um tempo aceitável.

Nas Subseções seguintes (2.3.2, 2.3.3, 2.3.4, 2.3.5 e 2.3.6) serão apresentados as técnicas utilizadas neste trabalho para construção de modelos preditivos para auxiliar no processo de configuração automática de parâmetros do Hadoop desenvolvido. Segundo (JOHANSSON; LOFSTROM; SONSTROD, 2011), frequentemente, as técnicas de mineração de dados utilizam os dados armazenados, a fim de construir modelos preditivos. Os modelos preditivos possuem algoritmos e técnicas que vêm da aprendizagem de máquina, ou seja, uma sub-área da Inteligência Artificial que visa criar aplicações capazes de aprender. A aprendizagem consiste em obter um modelo generalizado a partir de exemplos (etapa indutiva). Então, os modelos preditivos são aplicados em novos dados (etapa dedutiva). De um modo geral, visa-se que o número de classificações incorretas seja minimizado ou o número de classificações corretas seja maximizado.

2.3.2 Regressão Linear (LR)

A regressão linear é a equação que estima um valor esperado para y, possuindo os valores para x. A LR foi o primeiro tipo de análise de regressão a ser rigorosamente estudada e posteriormente usada em aplicações práticas. Isto aconteceu devido ao fato dos modelos que dependem linearmente dos seus parâmetros desconhecidos serem mais fáceis de encaixar do que os modelos que não são linearmente relacionados com o seus parâmetros. Nesse caso, os dados são modelados utilizando funções de previsão linear, e o modelo de parâmetros desconhecido é estimado a partir dos dados. Esses modelos são chamados de modelos lineares[4].

O modelo Linear é um termo muito geral de modelos estatísticos, que contém modelo de regressão linear, o modelo de análise de variância, o modelo de análise de covariância, modelo linear de mistura e assim por diante. No mundo real, existe este tipo de caso: duas variáveis aleatórias, tais como X e Y têm alguma relação de dependência. O valor de X pode determinar parcialmente o valor de Y, mas essa decisão não é precisa. O valor de Y contém duas partes. Uma dessas partes é em função de X. Em muitos casos, esta função é linear ou linear aproximada (MA et al., 2011).

Os parâmetros ideais de um modelo de regressão linear podem ser encontrados na pesquisa gradiente descendente, ou calculado exatamente (RUSSELL; NORVIG, 2009). Em resumo, a LR é uma abordagem para modelar a relação entre uma variável escalar dependente Y e uma ou mais variáveis independentes denominadas X. Quando se trata de apenas uma variável independente é chamado de regressão linear simples. Caso contrário, o processo é chamado de regressão linear múltipla.

Seja a seguinte situação: Uma investigação do efeito de diferentes quantidades de fertilizante na produção de grama em solo calcário, mostrada na Figura 10. Observa-se uma forte relação que segue linearmente. A variável dependente, aquilo que deseja-se predizer, localiza-se no eixo vertical. A variável independente localiza-se no eixo horizontal. A ideia básica na regressão linear é escolher a reta $y = a + bx$ que minimiza a soma de quadrados de desvios verticais dos pontos até a reta, para conseguir prever o valor que se quer.

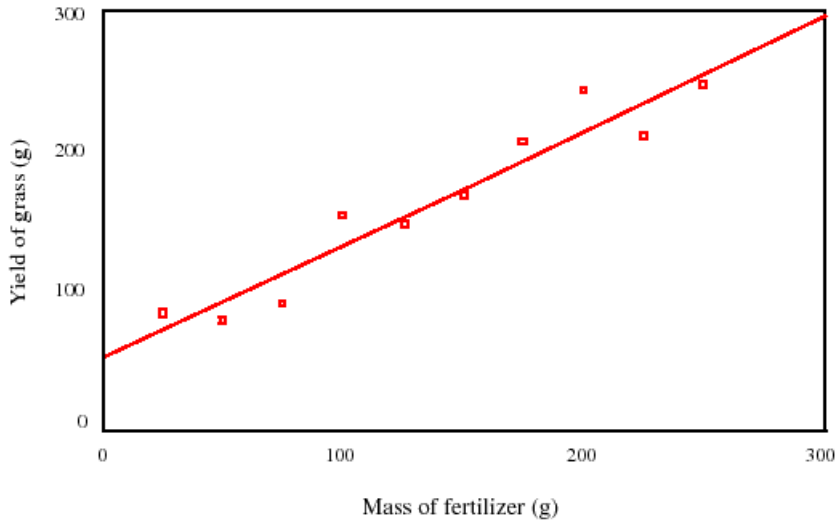
A Regressão linear só lida com relações lineares entre as variáveis dependentes e independentes. Ou seja, ela assume que existe uma relação linear entre elas, e nem sempre isso é válido.

2.3.3 Perceptron de Múltiplas Camadas (MLP)

Segundo Russell e Norvig (2009), uma rede neural é uma coleção de unidades ligadas entre si. As propriedades dessa rede são determinadas pela sua topologia e as propriedades dos neurônios. Uma rede neural é composta de unidades (mais conhecidas como neurônios) conectadas por arestas diretas que servem para propagar a ativação entre uma unidade

[4] https://en.wikipedia.org/wiki/Linear_regression

Figura 10 – Gráfico de dispersão de dados representando da regressão linear.

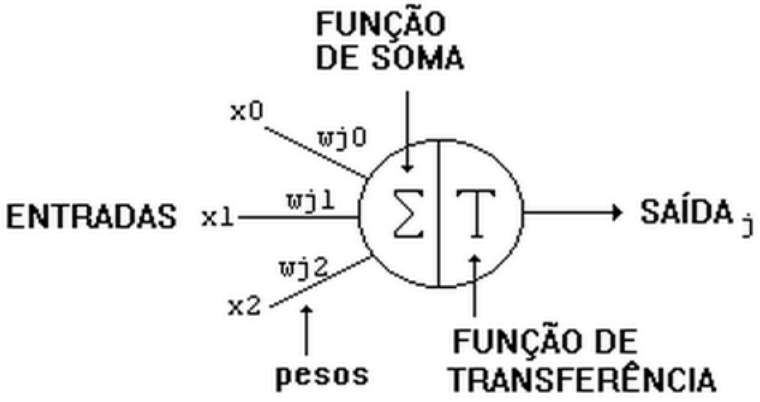


Fonte: SILVIA (2015)

e outra. Cada aresta tem um peso numérico associado a ela que representa a força da conexão. Em suma, a rede neural artificial é um sistema de neurônios ligados por conexões sinápticas e dividido em neurônios de entrada, que recebem estímulos do meio externo, neurônios internos e neurônios de saída, que se comunicam com o exterior.

O neurônio artificial, ver Figura 11, consiste em uma estrutura lógico-matemática que busca imitar a atuação de um neurônio biológico. Ele é composto por entradas, cujas ligações são realizadas através de pesos. Os estímulos que foram captados pelas entradas são processados pela função de soma ou ativação, e os valores obtidos são passados para função de transferência.

Figura 11 – Estrutura de um neurônio artificial.



Fonte: ARTIFICIAIS (1998)

Rosemblatt em 1957 mostrou em seu livro (Principles of Neurodynamics) o modelo dos "Perceptrons". Nesse modelo, os neurônios eram organizados em uma camada de entrada

e uma camada de saída, onde os pesos das conexões eram adaptados a fim de se atingir a eficiência sináptica, isto é, uma rede neural de duas camadas, usado no reconhecimento de caracteres (NEURAIS, 2000).

Rumelhart, Hinton e Williams em 1986 desenvolveram o algoritmo de treinamento "backpropagation" para treinar de maneira eficiente redes com camadas intermediárias. No treinamento com o algoritmo backpropagation, a rede realiza duas etapas. Primeiramente, é apresentado um padrão à camada de entrada da rede. A atividade resultante é realizada através da rede, camada por camada, até o momento em que a resposta seja produzida pela camada de saída. Na segunda etapa, a saída adquirida é comparada à saída que se deseja para esse padrão em particular. Caso não esteja correta, calcula-se o erro. O erro é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das camadas internas vão sendo ajustados conforme o erro é retropropagado.[5]

O modelo de rede neural mais comum é o Perceptron de Múltiplas Camadas (MLP) mostrada na Figura 12. Este tipo de rede neural é conhecida como uma rede supervisionada porque requer uma saída desejada para aprender. O Perceptron de Múltiplas Camadas foi concebido para resolver problemas mais complexos, os quais não poderiam ser resolvidos pelo modelo de neurônio básico. Nele, os neurônios são organizados em camada de entrada e saída, onde os pesos das conexões são adaptados a fim de alcançar a eficiência sináptica. Os neurônios internos são de suma importância na rede neural pois provou-se que sem estes torna-se impossível a resolução de problemas linearmente não separáveis (NEURAIS, 2000). Os poblemas linearmente separáveis são problemas de classificação de padrões. Para os problemas linearmente não separáveis, não é possível fazer a separação dos padrões através de uma superfície de decisão linear.

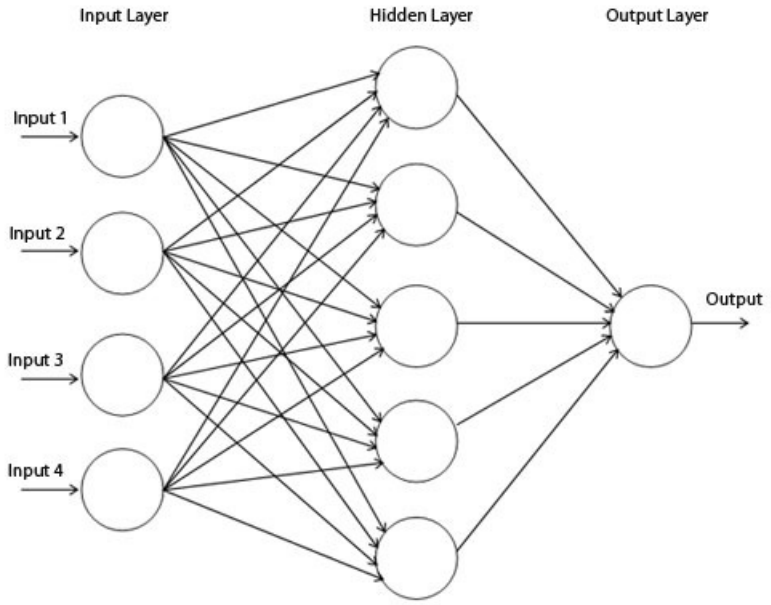
2.3.4 Redes de Funções de Base Radial (RBF)

De acordo com (XIAOFANG; JVLIN; GUODONG, 2010), a rede neural RBF é capaz de realizar aproximações universais. Essas utilizam funções radiais são uma classe de funções que podem ser empregadas em qualquer tipo de modelo (seja linear ou não linear) e qualquer tipo de rede. Veja sua estrutura na Figura 13.

> "A primeira camada é a conexão do modelo como o meio. A segunda camada ou camada escondida, realiza uma transformação não linear do espaço vetorial de entrada para um espaço vetorial interno que geralmente tem uma dimensão maior. A última camada, a camada de saída, transforma o espaço vetorial interno em uma saída, através de um processo linear. Os neurônios da camada escondida são funções radiais de base." (FERNANDES; A.D.D.; BEZERRA, 1999)

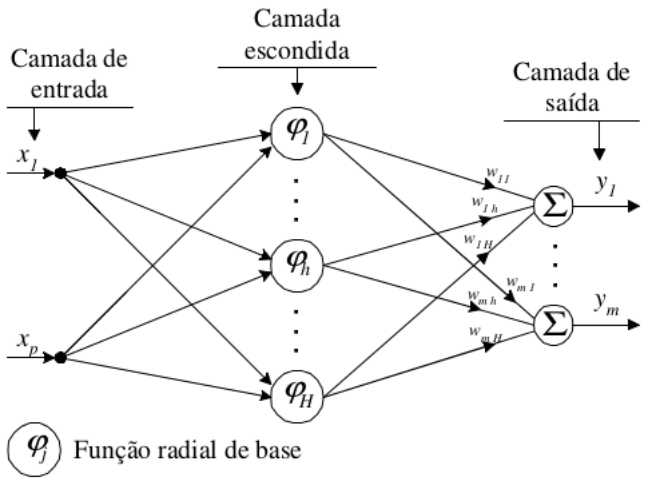
[5] http://www.icmc.usp.br/ andre/research/neural/MLP.htm

Figura 12 – Uma rede neural Perceptron de Múltiplas Camadas.



Fonte: SATIZÁBAL; PEREZ-URIBE (2014)

Figura 13 – Representação de uma rede neural de funções radiais de base.



Fonte: FERNANDES; A.D.D.; BEZERRA (1990)

Redes de Funções de Base Radial são redes *feed-forward* (onde as conexões entre os neurônios não formam um ciclo direcionado, isto é, alimentadas para frente) treinadas usando um algoritmo de treinamento supervisionado. A saída da rede é uma combinação linear de funções de base radial das entradas e parâmetros de neurônios. Redes de Funções de Base Radial têm muitos usos, incluindo a função de aproximação/estimação, classificação, modelagem de sistemas dinâmicos e séries temporais.

Essas redes tem como vantagem o fato de treinar muito mais rápido do que as redes de propagação de volta (do inglês *backpropagation*). O termo propagação de volta define

a maneira que a rede é treinada. Nesse caso, com o erro calculado, o algoritmo corrige os pesos em todas as camadas, partindo da saída até a entrada. Além disso, devido ao comportamento das unidades de função de base radial, as redes são menos suscetíveis a problemas com entradas não estacionárias (considerando que as entradas estacionárias tem a propriedade de que a média, variância e estrutura de autocorrelação não mudam no decorrer do tempo). Entretanto, (FERNANDES; A.D.D.; BEZERRA, 1999) afirma que uma grande desvantagem dessas redes está na maior exigência de memória.

2.3.5 Máquinas de Vetores de Suporte (SVM)

Máquinas de Vetores de Suporte (SVM) [6] compreendem um conjunto de métodos do aprendizado supervisionado que analisam os dados e reconhecem padrões, usadas para classificação e análise de regressão. São aplicadas a vários problemas de classificação de padrões e se popularizaram nos últimos anos.

Segundo (YIGITBASI et al., 2013), duas características inerentes às SVM têm motivado a explorá-lo para a modelagem de desempenho. Primeiramente, SVM é considerado um algoritmo de aprendizagem eficiente que tem uma boa capacidade de generalização e possui menos risco de sobreajuste comparado a outras abordagens. Ou seja, do inglês *overfitting*, o sobreajuste é quando o modelo estatístico se ajusta demais ao conjunto de dados ou amostra. Em segundo lugar, ao contrário de redes neurais artificiais, SVM é mais rápido para treinar.

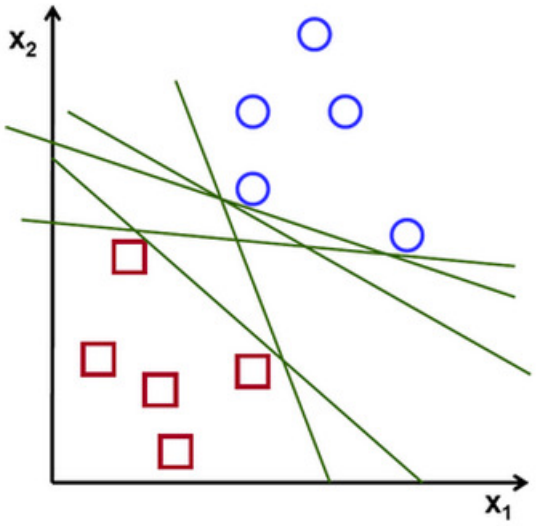
O SVM aplicado à regressão usa os mesmos princípios que o SVM aplicado à classificação. Em primeiro lugar, pelo fato da saída ser um número real (ao invés de uma classe nominal) torna-se muito difícil prever a informação manipulada, com infinitas possibilidades. No caso de regressão, uma margem de tolerância é definida em aproximação ao SVM.

Seja o seguinte problema, descrito em (OPENCV, 2011): Em um conjunto de pontos 2D linearmente separáveis que pertencem a uma das duas classes, encontrar uma linha reta que as separa. Na Figura 14, apresentam-se várias linhas para solucionar o problema, mas o objetivo é encontrar a linha que passa a partir de todos os pontos. Desse modo, o funcionamento do algoritmo SVM visa encontrar o hiperplano que dá a maior distância mínima para os exemplos de treino. Esta distância recebe o nome de margem. Veja a Figura 15.

Algumas das vantagens do SVM é, primeiramente, o fato de que ele consegue lidar bem com grandes conjuntos de amostras. Além disso, trata bem dados de alta dimensão. E por fim, o processo de classificação é rápido. As desvantagens residem, principalmente, no longo tempo de treinamento solicitado dependendo do número de amostras e dimensionalidade dos dados. Ademais, suas exigências de cálculo e armazenamento aumentam

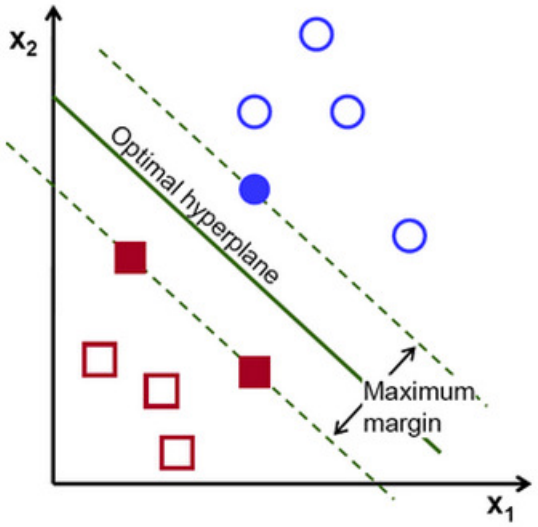
[6] Também conhecido como Regressão de Vetores de Suporte (SVR) no contexto da regressão.

Figura 14 – conjunto de pontos 2D linearmente separáveis.



Fonte: OPENCV (2011)

Figura 15 – conjunto de pontos 2D separados linearmento pelo hiperplano e margem.



Fonte: OPENCV (2011)

rapidamente com o número de vetores de treinamento (ZHANG; LI; YANG, 2005). O que pode aumentar o tempo de execução do algoritmo.

2.3.6 Processos Gaussianos (GP)

A regressão do processo Gaussiano nos permite prever os valores físicos em qualquer ponto, com um nível de incerteza prevista de forma eficiente. Os processos Gaussianos são considerados ferramentas poderosas para inferência. Eles são um método probabilístico, que podem fornecer estimativas da incerteza das previsões (ZHU; SUN, 2013).

Um processo Gaussiano é especificado por uma média e uma função de covariância.

A média é uma função de x, e a covariância é uma função de C(x, x') que expressa a covariância esperada entre o valor da função y nos pontos x e x'. A função real $y(x)$ em qualquer dado problema de modelagem é assumido como sendo uma única amostra dessa distribuição Gaussiana (LI; LI; LU, 2008). Segundo (LIAO; DATTA; WILLKE, 2013), os processos Gaussianos são comumente utilizados para implementações de configurações automáticas de diversos sistemas.

O processo Gaussiano é uma generalização de uma distribuição Gaussiana e tem a propriedade de marginalização (representa a densidade incondicional de x que pode ser calculada a partir da densidade conjunta por integração). GP controla as propriedades de dados aleatórios x por um processo aleatório f(x) e de forma síncrona descreve este processo aleatório por uma distribuição de probabilidade (LI; LI; LU, 2008).

Uma grande vantagem do uso dos processos gaussianos é que a previsão é probabilística para que se possa calcular intervalos de confiança empíricos e probabilidades de superação que poderiam ser utilizados para reaparelhar a previsão de alguma região de interesse. Além disso, os processos gaussianos são versáteis pois os modelos comuns são fornecidos, mas também é possível especificar modelos personalizados desde que sejam estacionários. Como desvantagem, os processos gaussianos perdem eficiência em espaços de alta dimensão, ou seja, quando o número de recursos excede algumas dezenas. Portanto, ele pode fornecer um desempenho ruim e perde eficiência computacional.[7]

[7] http://scikit-learn.org/stable/modules/gaussian_process.html

3 CONFIGURAÇÃO AUTOMÁTICA DO HADOOP

O Capítulo a seguir tem por função introduzir o problema da configuração automática do Hadoop e mostrar os trabalhos existentes na literatura que focam esse mesmo problema. A estrutura do Capítulo é a Seção de contextualização e a motivação para o estudo (3.1), e a exposição dos trabalhos relacionados, classificando-os em estáticos (3.2) e dinâmicos (3.3).

3.1 Contexto e Motivação

A grande quantidade de dados que precisam ser processados por instituições e governos é um dos principais desafios da computação atualmente. Mostra-se a seguir alguns contextos de manipulação de grandes quantidades de dados que existem atualmente.

- A importância de como lidar com grandes dados também está na ascensão das políticas de dados governamentais abertos. O trabalho de (SANTOS; SILVA, 2014) chamado IGOV, lida com uma grande quantidade de dados abertos brasileiros, desenvolvendo soluções para a integração de dados vindos de diferentes fontes e de maneira heterogênea. Desse modo, o trabalho discute uma solução para integrar dados vindos da saúde pública brasileira e do programa de transferência de renda Bolsa Família. Desse modo, apenas a liberação e produção de grandes dados não é condição suficiente para fazer análises mais aprofundadas, mostrando-se necessário o surgimento de tecnologias para o processamento e análise do grande volume de dados.

- Outro contexto consiste em explorar conjuntos de dados no contexto de mídia social, no intuito de ajudar os usuários de blogs a encontrar as informações significantes nessa vasta quantidade de dados que é a Web. O trabalho de (COSTA et al., 2012) propõe um arcabouço para a mineração desses dados, onde seu estudo de caso é uma aplicação de *e-commerce* para a análise de sentimento do usuário em relação a produtos específicos na Web. Nela, os dados encontrados em uma enorme quantidade não se apresentam necessariamente de maneira estruturada e homogênea. Desse modo, tornam-se necessárias soluções para ajudar no processamento dos dados (para futuras análises feitas por aplicações como a de (COSTA et al., 2012)).

Nas últimas décadas, a quantidade de dados gerados no mundo tem aumentado de maneira significativa. A Computação em Nuvem juntamente com o modelo de programação MapReduce, através do arcabouço Hadoop, têm sido utilizados para o processamento desses dados. Contudo, segundo (FILIERI; HOFFMANN; MAGGIO, 2014), a crescente

complexidade dos sistemas de computação está aumentando a dificuldade dos desenvolvedores e adminstradores de aplicações na nuvem. O que agrava ainda mais tal situação é a natureza dinâmica das aplicações distribuídas modernas assim como as peculiaridades de cada aplicação.

Assim, os desenvolvedores e administradores dessas aplicações têm dedicado cada vez mais tempo para a configuração ou ajuste de seus programas (TIWARI et al., 2009). Além disso, a dificuldade da configuração pode implicar em erros humanos além de ser inviável para aplicações com muitos parâmetros como aplicações MapReduce desenvolvidas com o Hadoop. Uma forma de resolver esse problema é através da configuração automática das aplicações para que elas sejam executadas com melhor proveito da infraestrutura de nuvem subjacente.

De acordo com (LAMA; ZHOU, 2012), não existe uma configuração única ideal do Hadoop que permita otimizar o desempenho de todos os tipos aplicações. Em geral, a otimização de parâmetros Hadoop requer um conhecimento específico do domínio da aplicação, um conhecimento aprofundado sobre o funcionamento interno do Hadoop e da infraestrutura de computação em nuvem de execução (ou clusters). Além disso, a configuração manual de parâmetros é um grande desafio devido às dependências (explícitas ou não) entre os parâmetros e os seus efeitos no sistema. Desse modo, o ajuste manual de parâmetros do Hadoop vem sido preterido principalmente pela imensidão do espaço de busca com centenas de configurações de parâmetros e pela complexidade de interação entre eles (LIAO; DATTA; WILLKE, 2013). Ou seja, tentar encontrar valores satisfatórios manualmente através de tentativa e erro tem um custo alto de tempo e esforço, além de não ser eficaz.

No intuito de auxiliar os programadores a configurar aplicações MapReduce, pesquisas nos últimos anos têm se dedicado à configuração automática de aplicações implementadas com o Hadoop. Especificamente, empregando técnicas para avaliar um conjunto de mapeamentos alternativos e selecionar aquele que obtiver o melhor desempenho (TIWARI et al., 2009). Dessa maneira, essas abordagens de configuração automática de parâmetros do Hadoop tentam encontrar valores ótimos para um conjunto de parâmetros de tal forma que o desempenho seja maximizado (LIU et al., 2012). Contudo, ainda existem poucos estudos relacionados a otimizações de desempenho em aplicações do Hadoop. Os trabalhos relacionados possuem limitações como o fato de serem, em geral, soluções intrusivas (ao contrário desse estudo proposto). Além disso, são estudos que apresentam uma única solução, diferente da abordagem proposta que expõe várias soluções, se mostrando extensível.

De uma maneira geral, as abordagens de configuração automática encontradas na literatura estão divididas em *configuração automática estática* e *configuração automática dinâmica*. As próximas Seções explicam cada abordagem e apresentam os trabalhos mais relevantes encontrados na literatura para cada uma dessas categorias.

3.2 Abordagens Estáticas

A *configuração automática estática*[8] *refere-se à configuração antes da aplicação ser executada*. A configuração estática do Hadoop é a mais utilizada atualmente, tendo em vista o estado da arte. Geralmente, ela é iniciada selecionando uma configuração com base no ajuste padrão ou através do conhecimento específico da aplicação. Depois são executados testes e são coletados dados referentes ao comportamento da aplicação. Por fim, são geradas novas configurações baseadas no desempenho das aplicações. Os pontos positivos da configuração estática consistem na maturidade da abordagem, por ser mais conhecida e estudada, na menor complexidade de implementação, quando comparadas à configuração automática dinâmica. Além disso, é possível ter um controle maior do processo de configuração antes e durante a execução.

Alguns pontos negativos podem ser vistos na configuração automática estática, como especificados por (LI et al., 2014a). Primeiramente, esse processo consome maior tempo pois é necessário que as aplicações sejam executadas previamente. Além disso, a configuração automática estática ajusta os parâmetros uma vez e usa essa configuração ao longo de todo ciclo de vida de um trabalho. Como as características de uma aplicação são dinâmicas, tal abordagem pode por vezes não se adaptar às variações causando degradação de desempenho.

Alguns trabalhos na literatura, classificados nessa categoria, se destacam e são apresentados a seguir.

Babu (2010) indica que a combinação do MapReduce e computação em nuvem é uma proposta atraente para as organizações (pois facilita para os desenvolvedores a criação de aplições que lide com grandes dados sem se preocupar com tolerância a falhas, paralelização, entre outras coisas). Entretanto, no MapReduce, uma série de parâmetros de ajuste tem que ser definidos por usuários ou administradores de sistema. Os usuários muitas vezes têm problemas de desempenho, porque eles não sabem como definir esses parâmetros, ou porque nem sequer sabem que existem esses parâmetros. Desse modo, o Starfish faz uso de técnicas para automatizar a configuração dos parâmetros de ajuste para os programas MapReduce. O objetivo é proporcionar um bom desempenho para programas MapReduce executados em grandes conjuntos de dados. Esse recurso pode melhorar a produtividade dos usuários que não possuem as habilidades necessárias para otimizar programas, devido à falta de familiaridade com MapReduce ou com os dados a serem processados. Nesse caso, é uma abordagem que usa modelos de desempenho baseados em custo financeiro junto com funções matemáticas. Isso é feito no intuito de tentar encontrar a melhor escolha de configuração de parâmetros de um trabalho MapReduce para aplicação como Terasort (que é o estudo de caso pertencente ao pacote de exemplos do Hadoop). Entretanto, comparando os modelos baseados em custo versus os modelos

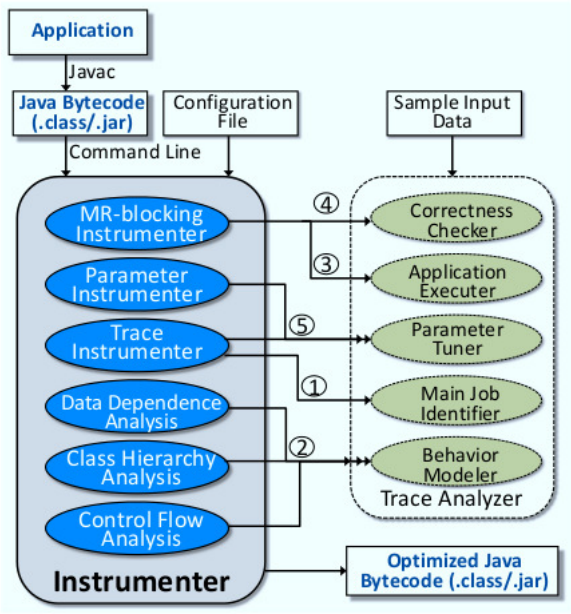
[8] Também chamada pela literatura como *offline* ou *tradicional*.

de aprendizagem de máquina, é percebido que é extremamente difícil para os modelos baseados nos custos simples predizer com precisão o desempenho de uma vasta gama de aplicações Hadoop. Isso acontece porque existe uma larga gama de grupos com CPU, armazenamento da memória e diferentes tecnologias de rede. Além disso, os modelos baseados em custos são estritamente vinculados a uma determinada versão do framework e não evoluem com ele.

O estudo feito por (LIU et al., 2012) apresenta o projeto e implementação de *Panacea*, um compilador que usa a combinação de análises estatísticas e busca exaustiva para otimizações gerais de desempenho em aplicações MapReduce baseadas em Hadoop. A entrada para Panacea é o código-fonte ou binário (bytecode) do aplicativo Hadoop. Na primeira rodada, o bytecode é instrumentado com instruções necessárias para a geração de rastreio. O *Trace Analyzer* é invocado e o aplicativo é executado com um ou mais conjuntos de dados de entrada de exemplo quando os dados da aplicação real não estão disponíveis. O *Trace Analyzer* identifica o trabalho principal no aplicativo MapReduce. Isto é feito através da comparação do tempo total de execução dos diferentes trabalhos e identifica aquela que executa com o tempo mais longo. Uma vez que o trabalho principal é identificado, verificações básicas de legalidade são executadas pelo *Instrumenter* para as duas otimizações, como a identificação de funções *map* e *reduce*. Se for bem-sucedido, na segunda rodada de instrumentação é aplicada com a finalidade de realização de checagens adicionais de legalidade em tempo de execução e o *Trace Analyzer* é chamado novamente. Isto combinado com a utilização de sinalizador aproximado indica que a aplicação segue o modelo de convergência iterativo. Se estiverem preenchidas as condições de legalidade, o *Instrumenter* aplica as otimizações e invoca o *Analyzer Trace* (este processo se repete algumas vezes para a configuração automática de parâmetros). O *Trace Analyzer* executa a aplicação otimizada e executa a análise de correção, se necessário, compara a saída do aplicativo otimizado com a da versão original. Se a análise de correção tiver êxito, o bytecode Java otimizado é a saída para um arquivo e uma mensagem é impressa informando o desenvolvedor do mesmo. A verificação foi realizada com a ajuda de cinco aplicações Hadoop comumente usadas (*K-means*, *K-core*, *PageRank*, *Wordcount*, *WCG*). Tal estudo possui como ponto forte a simplicidade da solução, porém, seu principal ponto fraco é que a utilização da busca exaustiva, o tempo de busca pode ser muito alto dependendo da extensão do espaço de busca das variáveis. Veja a arquitetura descrita na Figura 16.

Lama e Zhou (2012) propõem um sistema chamado AROMA que automatiza a alocação de recursos de nuvem heterogêneos. Ele permite a configuração automática de parâmetros do MapReduce baseadas em Hadoop no intuito de atingir as metas de qualidade de serviço e de minimizar os custos financeiros da alocação dos recursos. A arquitetura do AROMA é mostrada na Figura 17. Os usuários finais enviam tarefas como entrada para AROMA através de uma interface de linha de comando. O AROMA primeiro calcula o número e tipo de máquinas virtuais a serem alocadas baseado no binário da aplicação,

Figura 16 – Arquitetura do Panacea onde *Instrumenter* invoca diferentes componentes do *Trace Analyzer* na ordem mostrada (1-5).

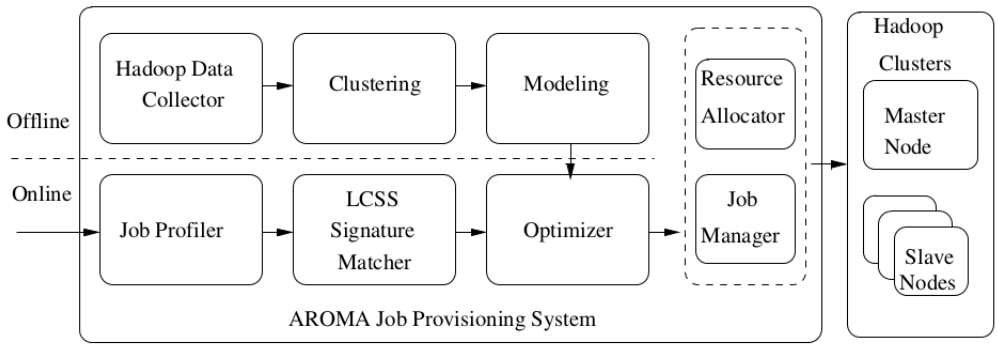


Fonte: LIU et al. (2012)

e atribui a parâmetros de configuração MapReduce adequadas para cumprir o prazo de conclusão a um custo mínimo. Então, realiza-se sua alocação de recursos em um grupo de máquinas virtuais inativas pré-instaladas com o software Hadoop. Finalmente, o *Job Manager* envia o trabalho ao nó mestre Hadoop, juntamente com seus parâmetros de configuração. Aqui, os principais desafios encontram-se na tomada de decisões rápidas e eficazes em tempo de submissão de trabalhos, alocar a quantidade certa de recursos e finamente ajustar as definições de configuração MapReduce para um trabalho com o tamanho de dados de entrada e prazo de conclusão imprevisíveis. Uma das contribuições é o provisionamento automatizado de trabalho do Hadoop em nuvem. Isso é feito para que os usuários finais possam utilizar os serviços de nuvem, e para isso não precisem adquirir um conhecimento aprofundado do sistema interno ou passando por otimização da configuração manual ineficaz, trabalhosa e demorada. Outra contribuição consiste em fornecer um mecanismo de provisionamento de trabalho geral que integra a alocação de recursos e configuração de parâmetros para atender a garantia de prazo de conclusão e para melhorar a eficiência de custos dos trabalhos MapReduce. Por fim, é capaz de tomar decisões eficazes de provisionamento para trabalhos, evitando um extenso e demorado *profiling* de trabalho. Ele faz isso através de técnicas inovadoras e práticas que combinam aprendizagem de máquina e otimização como *Support Vector Regression*(SVM) para treinamento, *k-medoid* para agrupamento de trabalhos passados armazenados e *Longest Common Subsequence*(LCSS) para estabelecer métricas de distância entre os agrupamentos.

O trabalho de (WANG; LIN; TANG, 2012) propõe o projeto *Predator* que é um oti-

Figura 17 – Arquitetura do AROMA como um sistema que integra a alocação de recursos e configuração de parâmetros para atender garantia prazo trabalho e melhorar a eficiência de custos de Hadoop MapReduce na Nuvem.

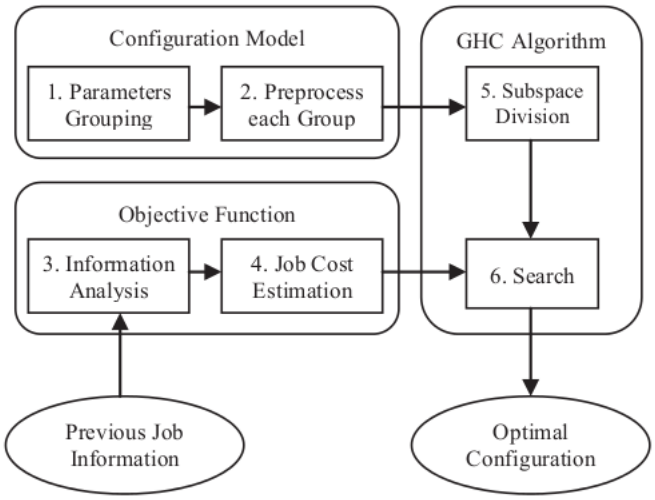


Fonte: LAMA; ZHOU (2012)

mizador de configuração do Hadoop que combina o ajuste de parâmetros com algoritmo de busca. Predator, cuja arquitetura é vista na Figura 18, seleciona parâmetros que tem o impacto mais significante no desempenho do trabalho MapReduce, obtendo sugestões para uma experiência prática de aprendizagem sobre o ajuste de parâmetros, onde cada experiência provê informação útil sobre parâmetros e faixa de valores para configuração ótima. Isso é realizado pré-processando esses parâmetros de acordo com o modelo de configuração e então usando algoritmo de busca para encontrar a configuração ótima baseada numa função objetivo. Então, primeiramente é estabelecido um modelo de configuração para classificar os parâmetros em quatro grupos de acordo com a sua capacidade de ajuste. Depois disso, acontece o pré-processamento de todos parâmetros em cada grupo. Após esse passo, adquire informações prévias sobre a execução do trabalho e analisa essas informações para obter propriedades do trabalho que sejam relevantes para o desempenho. Então, baseado no modelo de custo e nessas propriedades do trabalho, constrói-se trabalhos hipotéticos com diferentes configurações para os pontos no espaço de parâmetros e obtém-se o tempo de execução de cada trabalho como a função objetivo. Divide-se o espaço de parâmetros em sub-espaços iguais. Por fim, aplica-se o algoritmo *hill climbing* para buscar pelos valores de configuração ótima na base da função objetivo.

Wu e Gokhale (2013) apresentam o *PPABS*: um arcabouço que automatiza a configuração do Hadoop baseado em requisitos de desempenho da aplicação deduzida. A arquitetura do PPABS é mostrada na Figura 19. Nessa abordagem há a gestão de configuração automática de um cluster Hadoop baseada numa fase de aprendizagem automática que é usado para auto-ajuste. A fase de aprendizado de máquina requer treinamento. Com base no conhecimento adquirido, o sistema faz decisões de configuração eficazes para um trabalho de entrada e aplica essas decisões de configuração para automatizar todo o processo. Isso irá diminuir os encargos sobre os desenvolvedores de aplicativos. O PPABS é divido em duas fases distintas. A primeira fase chamada *Analyzer* treina o PPABS

Figura 18 – Arquitetura do Predator que tem a ideia básica de pré-processar esses parâmetros de acordo com o modelo de configuração e, em seguida, usar o algoritmo de busca para encontrar a configuração ideal com base na função objetivo.

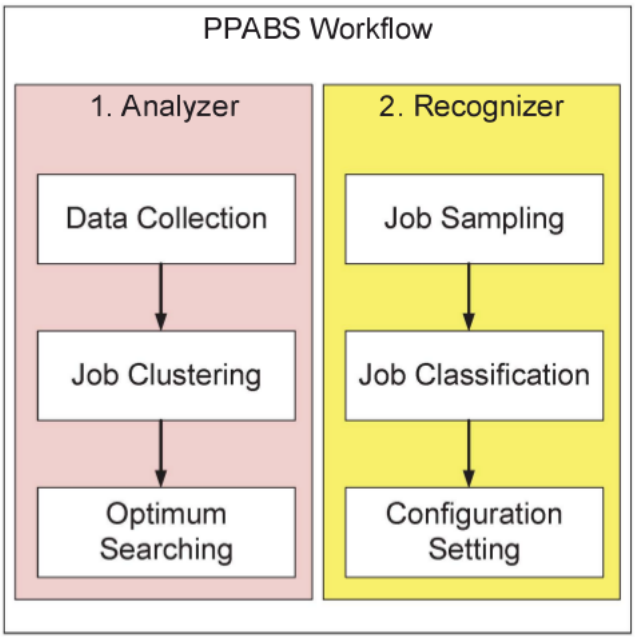


Fonte: WANG; LIN; TANG (2012)

para formar um conjunto de classes de equivalência de aplicações MapReduce para que a configuração de parâmetros mais apropriada seja determinada. Além disso, fornece um algoritmo de agrupamento k-means++ modificado e, adicionalmente, realiza modificações também no algoritmo *Simulated Annealing* para encontrar a solução desejada. A segunda fase nomeada *Recognizer* classifica a entrada de um trabalho desconhecida para essa classe de equivalência de modo que seus parâmetros de configuração possam ser configurados automaticamente. Ele classifica essa entrada de um trabalho desconhecida para uma das classes equivalentes pela primeira execução do trabalho em pequena parte da sua entrada usando configurações *default* do Hadoop e aplicando técnicas padrão de cognição para classificá-los.

Yigitbasi et al. (2013) propõem uma abordagem para tornar o processo de configuração de parâmetros mais efetivo. Nesse trabalho, os autores exploram modelos de desempenho baseados em aprendizagem de máquina com diversas aplicações MapReduce e configuração de cluster, e mostram que o modelo *Support Vector Regression* (SVR) teve uma boa precisão e também é computacionalmente eficiente. Para eleger tal abordagem, os autores comparam algumas abordagens para criação de modelos baseados em aprendizagem de máquina baseadas em regressão: *Simple Multiple Linear Regression*(MLR), *Multiple Linear Regression with Parameter Interactions*(MLR-I), *Multiple Linear Regression with Quadratic Effects*(MLR-Q), *Multiple Linear Regression with Parameter Interactions and Quadratic Effects*(MLR-IQ), *Artificial Neural Networks*(ANN), *Model Tree*(M5Tree) e *Support Vector Regression* (SVR). Eles desenvolvem alguns modelos de desempenho baseados em aprendizagem de máquina para dois *benchmarks* do Hadoop (*Wordcount* e *Sort*). Foram usados dados coletados de duas diferentes configurações de cluster, e avaliou-

Figura 19 – Arquitetura do PPABS que pode ser visto como duas partes principais: o *Analyzer* e o *Recognizer*. O *Analyzer* é baseado na passado e é um passo off-line; o *Recognizer* no entanto executa cada vez que um cliente envia um novo trabalho e é semi-online



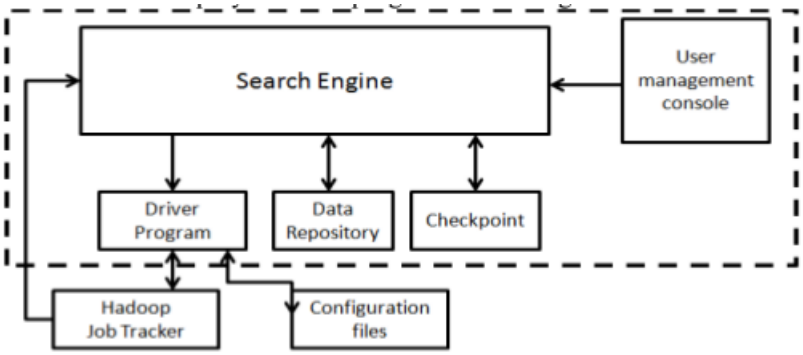
Fonte: WU; GOKHALE (2013)

se esses modelos em termos de precisão, desempenho computacional, e a sua sensibilidade ao tamanho dos dados de treinamento. Então, o estudo propõe a abordagem de configuração automática baseada em aprendizagem de máquina que usa algoritmos de busca inteligente para treinar os modelos de desempenho e explorar o espaço de parâmetros.

O projeto Gunther, proposto por (LIAO; DATTA; WILLKE, 2013), avalia abordagens para a configuração automática do Hadoop MapReduce, incluindo uma baseada em custo, a Starfish (BABU, 2010), e modelos de aprendizagem de máquina. O Gunther utiliza um Algoritmo Genético que é especialmente projetado para identificar definições de parâmetros agressivamente que resultam em melhores tempos de execução dos trabalhos. O Gunther é avaliado em dois tipos de clusters, cada um com características diferentes. Os experimentos demonstram que Gunther pode obter um desempenho próximo do ideal dentro de um pequeno número de tentativas. Em contraste com os modelos baseados em custos, os modelos de aprendizagem de máquina definem coeficientes de aprendizado do modelo e, portanto, são mais adaptáveis e flexíveis. Infelizmente, é difícil para a construção de um modelo preciso de aprendizado de máquina sem um grande conjunto de treinamento, envolvendo centenas ou potencialmente milhares de tentativas. Eles avaliam as abordagens de configuração automática utilizando as duas métricas: (i) *eficiência*, ou o quão rápido a busca pode encontrar uma boa configuração, e (ii) *eficácia*, que mede a melhoria do desempenho alcançado. As outras abordagens de algoritmos de busca global comparadas que elegeram o Algoritmo Genético (AG) como melhor alternativa foram:

simulated annealing (SA), *particle swarm optimization* (PSO) e *recursive random search* (RRS) (LIAO; DATTA; WILLKE, 2013). Os estudos de casos que são utilizados nesse estudo foram Sort, Kmeans,TeraSort e Nutch. Veja a arquitetura na Figura 20.

Figura 20 – Arquitetura do Gunther que consiste em um algoritmo de busca implementado no motor de busca (MB). MB gera uma nova configuração e pede ao programa para executar o aplicativo através do Hadoop *JobTracker* com a nova configuração. Após a execução é completa, SE escreve arquivos de log para o repositório e analisa-los a obter tempos de execução. A busca termina após o algoritmo de satisfazer os critérios de convergência ou chega a um número especificado de tentativas.



Fonte: LIAO; DATTA; WILLKE (2013)

3.3 Abordagens Dinâmicas

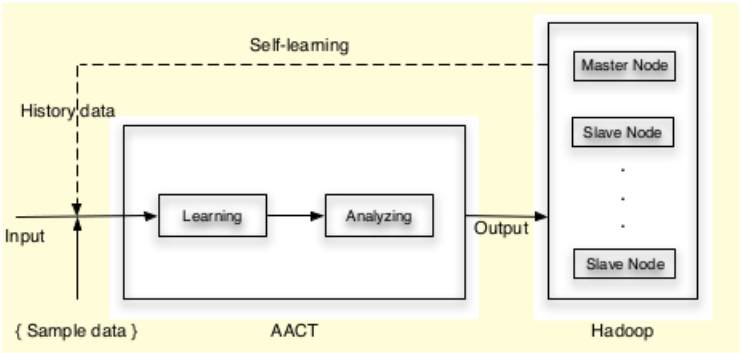
A *configuração automática dinâmica é realizada em tempo de execução da aplicação*, isto é, ajusta os parâmetros do Hadoop diversas vezes ao longo do ciclo de vida de um trabalho. Essa abordagem é mais adequada para cenários que acompanham o dinamismo inerentes aos trabalhos MapReduce. A primeira desvantagem desse tipo de abordagem é o alto grau de complexidade de implementação, inclusive alguns parâmetros são ainda mais difíceis de manipular dinamicamente que outros. Outra desvantagem da configuração automática dinâmica é que o funcionamento de determinados parâmetros pode influenciar diretamente outros parâmetros. Então, quando se manipula esses parâmetros em tempo de execução, não há um controle total de como a aplicação irá se comportar, já que o impacto pode não ser necessariamente local na execução. Ou seja, esses ajustes podem anular ou atenuar o efeito de outros parâmetros ou maximizar o efeito de parâmetros que não se desejava.

Os trabalhos a seguir apresentam soluções para configuração de parâmetros do Hadoop dinamicamente.

Em (LI et al., 2014), os autores propõem o AACT, uma ferramenta para otimizar o desempenho do sistema através da aprendizagem da relação interna entre os fatores de impacto. Sua arquitetura é mostrada na Figura 21. O AACT aprende relação entre os parâmetros de configuração, recursos de hardware, características do programa e o

desempenho do sistema, e fornece uma configuração melhor, analisando a função. Por outro lado, os novos resultados também serão recolhidos periodicamente como uma nova amostra para melhorias de precisão de aprendizagem adaptativa. O componente que coleta dados extrai o tempo de execução, o tamanho de dados de entrada e parâmetros de configuração de vários trabalhos passados do Hadoop *JobTracker*. Entretanto, um problema neste trabalho é a precisão e eficiência em aprender e, posteriormente, ajustar os parâmetros dinamicamente. Dado um aplicativo para ser executado em uma determinada plataforma, AACT procura automaticamente as configurações do sistema otimizado a partir de configurações candidatos. O AACT analisa a relação entre as configurações influentes e desempenho do sistema através da concepção de um modelo matemático e, em seguida, ajusta dinamicamente os parâmetros para alcançar otimização de desempenho. Depois de treinar o modelo na plataforma de destino, AACT aprende o relacionamento interno e faz previsões em conformidade. Com base na previsão, AACT poderia recomendar uma configuração otimizada e ajustá-las de forma dinâmica. Novos dados coletados serão aprendidos por AACT para melhorar a precisão da recomendação. AACT aplicado ao Hadoop mostra que o Hadoop é capaz de se adaptar às configurações para o sistema de forma dinâmica e conduzir o sistema para uma configuração ótima.

Figura 21 – Arquitetura do AACT aprende a relação entre a função de parâmetros de configuração, recursos de hardware, características do programa e o desempenho do sistema; em seguida, fornecer uma configuração ideal para alcançar a otimização.

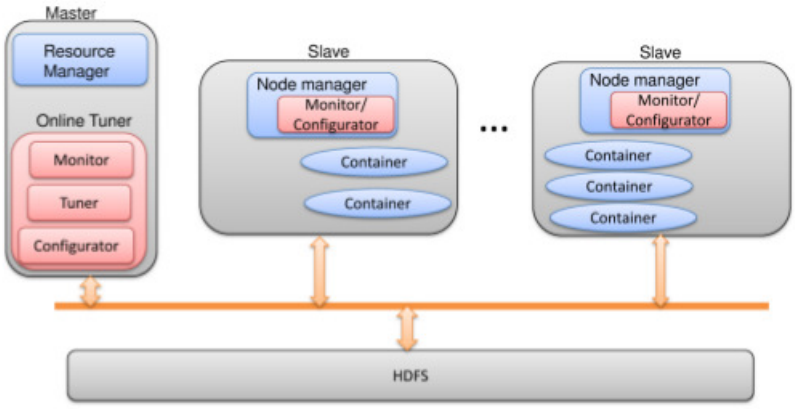


Fonte: LI et al. (2014a)

Por fim, o trabalho de (LI et al., 2014a) explora sistematicamente o espaço de parâmetros do Hadoop e seleciona uma configuração sub-ótima. Os autores argumentam que os ajustes estáticos existentes são lentos e pouco eficientes, já que ocasionam muitas execuções de teste e esforço humano significativo. Além disso, os autores defendem a ideia de que nenhuma configuração é adequada para todas as tarefas dentro de um trabalho. Ou seja, o ajuste tradicional estático configura os parâmetros de uma vez e usa a configuração em todo o ciclo de vida de um trabalho. Assim, as características do trabalho e a utilização dos clusters são dinâmicas e o ajuste estático pode, certas vezes,

não adaptar-se a tais variações, podendo causar degradações de desempenho. Para esse problema, os autores propõem um sistema de ajuste de desempenho dinâmico chamado *MROnline*, cuja arquitetura é mostrada na Figura 22. O *MROnline* monitora a execução dos trabalhos, ajusta parâmetros associados ao desempenho baseado em estatísticas coletadas e fornece controle de baixa granuladidade sobre a configuração de parâmetros. Para encontrar uma solução sub-ótima, o *MROnline* utiliza um algoritmo *hill climbing* baseado em caixa-cinza (onde parte detalhes técnicos do Hadoop ou aplicação devem ser acessíveis) para, sistematicamente, buscar através do espaço e encontrar a configuração desejada. O trabalho concentra-se em parâmetros-chave que afetam as tarefas em tempo de execução. Os autores estabelecem como trabalhos futuros investigar ajuste de parâmetros tais como o número de *mappers*, o número de *reducers* e o valor de *slow start*.

Figura 22 – Arquitetura do MROnline que é um sistema de ajuste de desempenho on-line que monitora a execução de um trabalho, atribuindo valores aos parâmetros de desempenho com base em estatísticas recolhidas. Permite que cada tarefa tenha uma configuração diferente, em vez de ter de usar a mesma configuração de todas as tarefas.



Fonte: LI et al. (2014b)

4 UMA ABORDAGEM NÃO INTRUSIVA PARA CONFIGURAÇÃO AUTOMÁTICA DE APLICAÇÕES HADOOP (ANICAH)

Essa Seção introduz e descreve as contribuições desse trabalho em três partes. A primeira parte expõe os objetivos pretendidos por esse trabalho (4.1). Depois, a Seção 4.2 descreve a arquitetura da abordagem proposta para configuração automática de parâmetros do Hadoop. Por fim, a Seção 4.3 explica como foi realizada a implementação do protótipo real da abordagem.

4.1 Objetivos

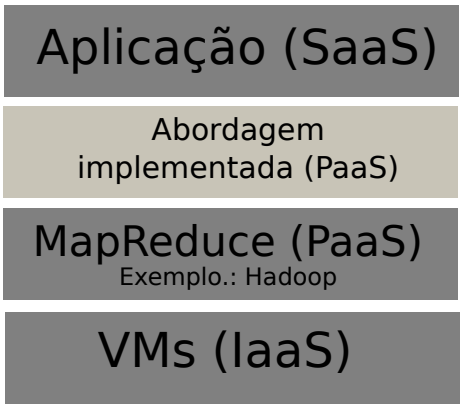
Como foi discutido no Capítulo 3, a configuração adequada do Hadoop permite melhorar o desempenho de suas aplicações. Contudo, a grande quantidade de parâmetros de configuração torna difícil saber *(i)* quais parâmetros variar; *(ii)* qual faixa de valores no espaço de busca podem melhorar o desempenho; assim como *(iii)* obter precisão de configuração. **O objetivo geral desse trabalho é propor uma solução para melhorar o tempo de execução de aplicações Hadoop através da configuração automática de parâmetros**. Especificamente, esse trabalho foca na *configuração automática estática*, ou seja, na configuração antes das aplicações serem executadas (ver Seção 3.2 para mais detalhes). A configuração automática facilita o desenvolvimento e a gestão de aplicações Hadoop, poupando desenvolvedores e administradores de sistemas de lidar com detalhes de configuração de arcabouços Hadoop.

Além disso, tem-se também como objetivo propor uma abordagem *transparente* e *independente* em nível de plataforma como serviço (PaaS) da computação em nuvem, ver Figura 23. Por transparência, entende-se que a solução não seja intrusiva aos desenvolvedores de aplicações Hadoop (que não precisa necessariamente ser um SaaS), ou seja, que os desenvolvedores não tenham que utilizar bibliotecas externas e que nenhuma modificação seja feita nas aplicações. Por independência, entende-se que a solução não requeira a modificação do Hadoop, nem no seu código-fonte nem como extensões (*plug-ins*). Assim, a solução para configuração automática deve lidar com o problema como sendo caixa-preta, onde não se tem acesso ao código-fonte tanto da aplicação quanto do Hadoop.

O Hadoop (HADOOP, 2008) foi escolhido por ser o arcabouço software livre[9] mais robusto e completo para aplicações Mapreduce. A configuração automática estática é preferida em relação à configuração dinâmica porque não intervém durante a execução da aplicação, logo não degrada o seu desempenho. Além disso, a implementação da configuração estática é mais simples, logo, além da implementação não ser complexa, facilita a evolução e manutenção do seu código.

[9] Nesse caso, o fato de o Hadoop estar sob uma licença de *software livre* é importante apenas para poder utilizá-lo sem restrições, uma vez que a abordagem não requer acesso ao código fonte.

Figura 23 – A contribuição desse trabalho situa-se na camada *Platform-as-a-Service* da computação em nuvem, atuando como um complemento ao Hadoop. O objetivo é melhorar o desempenho de aplicações Hadoop através da configuração automática de parâmetros.



Fonte: Elaborada pelo autor. (2015)

A Tabela 1 compila os trabalhos mais relevantes encontrados na literatura para configurações automáticas estática e dinâmica do Hadoop, comparando-os com a contribuição desse trabalho. Observa-se que técnicas de Aprendizagem de Máquina são utilizadas como soluções para a busca da melhor configuração automática de parâmetros do Hadoop. Alguns trabalhos relacionados, tais como (YIGITBASI et al., 2013) e (LIU et al., 2012), utilizam a busca exaustiva, uma maneira mais rudimentar de encontrar a configuração melhor para as aplicações. Assim, essas soluções levam muito tempo para encontrar a configuração adequada a depender da extensão do espaço de busca. Outros trabalhos adotam técnicas mais complexas com o uso da aprendizagem de máquina para encontrar configurações com melhor desempenho (LAMA; ZHOU, 2012), (WANG; LIN; TANG, 2012), (WU; GOKHALE, 2013), (BABU, 2010), (LIAO; DATTA; WILLKE, 2013), (LI et al., 2014) e (LI et al., 2014a). Dentre essas técnicas, destacam-se a *Support Vector Machine* (SVM), *Hill climbing*, *Simulated Annealing* e Algoritmos Genéticos. Pode-se observar que eles manipulam uma grande quantidade de parâmetros simultaneamente, o que torna o processo de busca mais complexo e demorado. O Gunther (LIAO; DATTA; WILLKE, 2013) apresenta bons resultados na literatura para o problema de configuração automática do Hadoop utilizando Algoritmos Genéticos. O Gunther mostra bons resultados comparando com outras técnicas de busca e com um número aceitável de iteraçãoes.

Em contrapartida, o trabalho proposto intitulado *Abordagem Não Intrusiva para Configuração Automática para Hadoop* (ANICAH) utiliza apenas três importantes parâmetros de configuração do Hadoop, segundo um levantamento do estado da arte: o número de *mappers*, o número de *reducers* e o valor de *slowstart*. Consequentemente, a contribuição apresenta uma solução simples e utiliza os parâmetros que impactam no desempenho das aplicações do Hadoop. Além disso, o trabalho proposto utiliza Algoritmos Genéticos combinado com Modelos preditivos criados através das seguintes técnicas: Redes de Funções

Tabela 1 – Comparativo entre os trabalhos na literatura que propõem a configuração automática do Hadoop. A contribuição destaca-se pela combinação de Algoritmos Genéticos com técnicas de aprendizado de máquina.

Abordagem	**Tipo de configuração**	**Ano**	**Abordagem empregada**	**N. de parâmetros**	`mappers`	`reducers`	`slow start`
Starfish (YIGITBASI et al., 2013)	estático	2010	Modelos de negócios	12	*block size*	sim	não
PANACEA (LIU et al., 2012)	estático	2012	Busca exaustiva (BE)	8	*map tasks*	sim	não
AROMA (LAMA; ZHOU, 2012)	estático	2012	Máquinas de Vetores de Suporte (SVM), *k-medoid* e LCSS	9	não	sim	não
PREDATOR (WANG; LIN; TANG, 2012)	estático	2012	*Hill climbing* (HC)	até 23	não	sim	sim
PPABS (WU; GOKHALE, 2013)	estático	2013	k-means++ e *Simulated Annealing* (AS)	8	*block size*	não	não
TOWARDS (BABU, 2010)	estático	2013	Máquinas de Vetores de Suporte (SVM)	5	não	sim	não
GUNTHER (LIAO; DATTA; WILLKE, 2013)	estático	2013	Algoritmo Genético (AG)	6	não	sim	não
AACT (LI et al., 2014)	dinâmico	2014	modelo matemático e aprendizagem	10	*block size*	sim	não
MRONLINE (LI et al., 2014a)	dinâmico	2014	*Hill Climbing* (HC)	13	não	não	não
Abordagem Não Intrusiva para Configuração Automática para Hadoop (ANICAH)	**estático**	**2015**	***A*lgoritmos Genéticos (AG) e Modelos preditivos**	**3**	**split size e job maps**	**sim**	**sim**

Fonte: Elaborada pelo autor. (2015)

de Base Radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP) e Perceptron de Múltiplas Camadas (MLP).

4.2 Arquitetura

A fim de alcançar os objetivos descritos na Seção 4.1, esse trabalho propõe uma arquitetura para configuração automática de aplicações MapReduce que utilizam o Hadoop. Essa arquitetura melhora o desempenho do Hadoop de maneira transparente, não requer acesso aos códigos fonte de aplicações e nem do Hadoop.

A Figura 24 ilustra a execução de uma aplicação Hadoop e a arquitetura proposta. Como falado anteriormente, ao utilizar o Hadoop, o programador deve implementar sua aplicação MapReduce utilizando bibliotecas do Hadoop. Por exemplo, uma aplicação escrita em Java e que é compilada (`javac`) gerando um arquivo bytecode empacotado (`.jar`). Para executar a aplicação, é necessário indicar qual é a entrada da aplicação Hadoop, que será copiada para o HDFS, o sistema de arquivo do Hadoop. Em seguida, executa-se o binário do Hadoop passando o `.jar` e sua entrada como parâmetro. Até então, esse é o ciclo padrão de desenvolvimento e execução de uma aplicação no Hadoop.

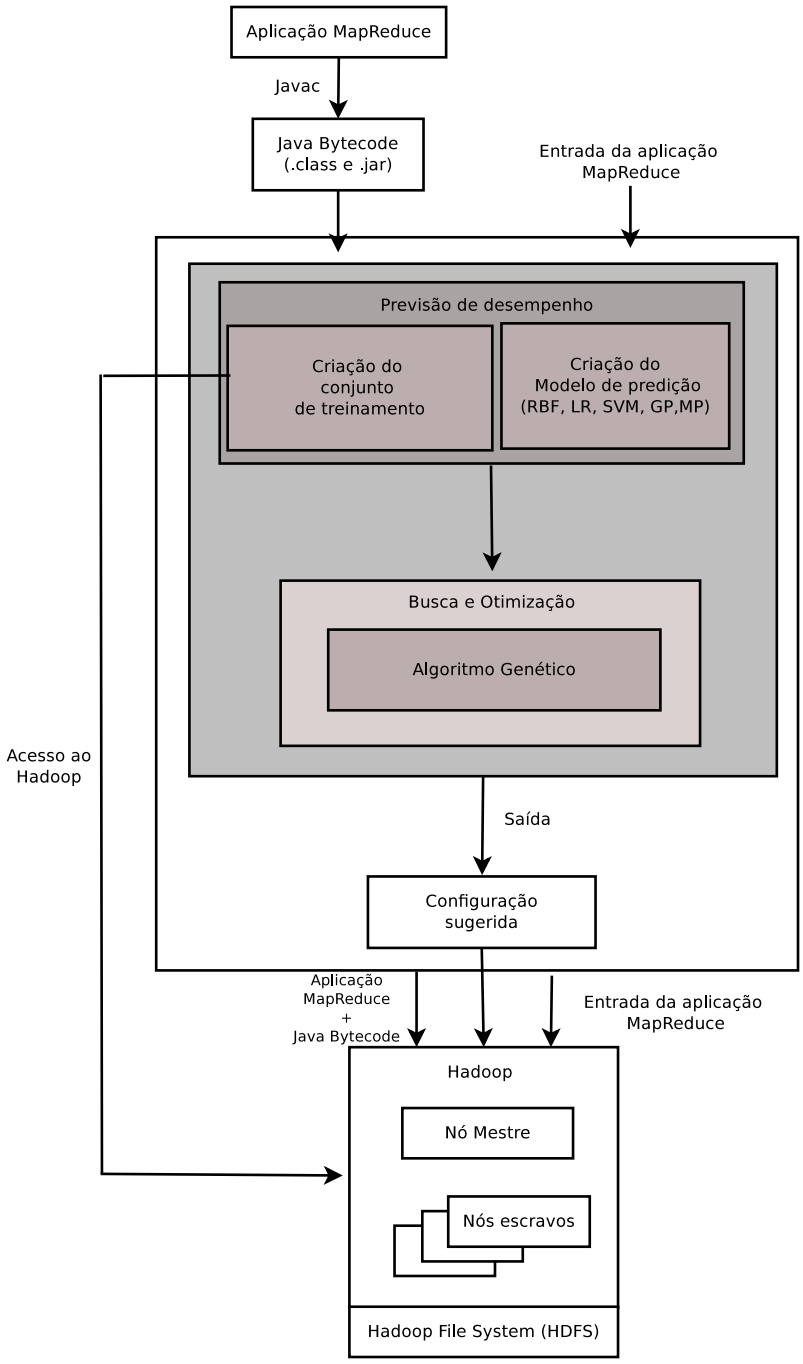
A arquitetura da Abordagem Não Intrusiva para Configuração Automática para Hadoop (ANICAH) situa-se no meio do ciclo de *execução* da aplicação, ou seja, não interfere no processo de desenvolvimento da aplicação. ANICAH atua como uma solução de configuração automática do Hadoop para essa aplicação. Além disso, a ANICAH utiliza Modelos prreditivos associada a Algoritmos Genéticos para a escolha dos parâmetros. A arquitetura da ANICAH consiste nos componentes chave descritos a seguir (ver Figura 24):

Previsão de desempenho Esse módulo é muito importante e responsável pela criação dos modelos de predição modulares que podem ser escolhidos dependendo do contexto e da aplicação. Esse módulo é dividido em *Criação do conjunto de treinamento* e *Criação do Modelo de predição*.

Criação do conjunto de treinamento Na *criação do conjunto de treinamento*, os tempos de execução para diferentes configurações do Hadoop são registrados através de execução real do Hadoop. Essas configurações são combinações dos seguintes parâmetros do Hadoop: número de *mappers*, número de *reducers* e valor de *slow start*. Esses parâmetros foram escolhidos por serem parâmetros importantes que influenciam no desempenho das aplicações MapReduce (LI et al., 2014a). Nesse caso, o MROnline (LI et al., 2014a) não usa esses parâmetros pois utiliza a configuração automática dinâmica e tais parâmetros são difíceis de serem manipulados dinamicamente, especificando isso como trabalho futuro. Além disso, o projeto da arquitetura utiliza a abordagem de poucos parâmetros para poder reduzir o espaço de busca, em vez de utilizar vários parâmetros e inviabilizar buscas mais extensas por configurações [10]. O conjunto

[10] De acordo com (LIAO; DATTA; WILLKE, 2013), a eficiência da busca pode ser melhorada com um menor impacto sobre a eficácia através da redução da dimensionalidade do espaço de busca, isto é, o número de parâmetros a serem manipulados e buscados.

Figura 24 – Visão geral da Arquitetura do ANICAH.



Fonte: Elaborada pelo autor. (2015)

de treinamento criado nessa fase é fundamental para a etapa de predição de desempenho, pois o conjunto de treinamento é a entrada para criação dos modelos que irão prever os tempos de execução. Tendo sido criado esse conjunto, ele é repassado para a criação do modelo preditivo.

Criação do Modelo de predição É criado através dos modelos de regressão gerados pelas seguintes técnicas de aprendizagem de máquina, utilizadas de maneira modular: Redes de Funções de Base Radial (RBF), Regressão Linear

(LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP) e Perceptron de Múltiplas Camadas (MLP). Tais modelos de regressão são utilizadas para mensurar, individualmente, o tempo de execução de uma aplicação Hadoop dada uma configuração a ser investigada. Essas técnicas utilizam o conjunto de treinamento descrito anteriormente para gerar um modelo que preveja o tempo de execução das configurações sem necessariamente executá-las, apenas a partir do comportamento das instâncias de treinamento. O módulo de previsão de desempenho pode ser invocado dentro o Algoritmo Genético para que os valores previstos pelo modelo preditivo sejam atribuídos à taxa de aptidão da configuração. Desse modo, ao final da execução do Algoritmo Genético, obtêm-se uma configuração a ser sugerida. Essa configuração encontrada é passada para que o Hadoop execute a aplicação para aqueles valores de parâmetros.

Busca e Otimização Tal módulo é responsável pelo processo de buscar uma boa configuração que reduza o tempo de execução das aplicações Hadoop, otimizando o desempenho. Isso é realizado na execução do *Algoritmo Genético*.

Algoritmo Genético O componente *Algoritmo Genético* é responsável por encontrar a configuração de parâmetros relativa aos menores tempos de execução. O Algoritmo Genético foi escolhido baseado nos resultados de Liao, Datta e Willke (2013) que afirmam que o Algoritmo Genético obteve um desempenho melhor e define coeficientes de aprendizado do modelo, sendo mais adaptáveis e flexíveis [11]. A ANICAH utiliza um Algoritmo Genético Modificado descrito no Algoritmo 2. Nesse algoritmo, cada configuração é equivalente a um indivíduo onde seu genótipo é dado da seguinte maneira: *(número de mappers, número de reducers, valor de slowstart)*. Além disso, a taxa de aptidão é o tempo de execução da aplicação para tal configuração, que pode ser previsto pelo modelo preditivo. É conveniente lembrar que é estabelecido e especificado um espaço de busca para que o Algoritmo Genético possa atuar e fornecer configurações pertencentes a esse espaço.

Desse modo, o AG gera uma população inicial onde as configurações são sempre pertencentes ao espaço de busca. Após isso, é determinada a quantidade de gerações que serão formadas ao longo da execução do algoritmo. Posteriormente, cada indivíduo sofrerá uma avaliação com base na taxa de aptidão encontrada através do modelo de predição. Isso acontece para buscar quais indivíduos possuem o menor tempo de execução no Hadoop. Em seguida, é selecionada a próxima população a partir da população anterior que possui indivíduos com

[11] O Algoritmo Genético foi mais eficiente quando comparado às abordagens *simulated annealing* (SA), *particle swarm optimization* (PSO) e *recursive random search* (RRS) (LIAO; DATTA; WILLKE, 2013).

melhor aptidão. A seguir, os indivíduos são cruzados para produzir filhos que podem ou não gerar tempos de execução no Hadoop que sejam menores. Após a geração de outras configurações pelo cruzamento, é feita a mutação para inserir características novas na população que pode melhorar ou piorar a solução para o problema. Por fim, atualiza-se a população que será avaliada e passada para a seleção da nova população quantas vezes foram especificadas *a priori*.

Algoritmo 2: ALGORITMO GENÉTICO MODIFICADO

```
1  Seja P(t) uma população da geração t
2  t ← 0
3  Inicializa aleatoriamente a polulação P(t).
4  enquanto t < n_de_gerações faça
5      populacao ← 0
6      enquanto populacao < tamanho_populacao faça
7          taxa_de_aptidao ← Previsao_de_desempenho
8          Avalia a população P(t)
9          populacao ← populacao + 1
10     fim
11     t ← t + 1
12     Seleciona-se a população atual P(t) a partir da população anterior P(t-1).
13     Aplicar-se o cruzamento (ou crossover) sobre a população P(t).
14     Realiza-se e a mutação dos indivíduos resultantes do cruzamento.
15     Atualiza a população P(t).
16 fim
```

Com relação à escolha dos parâmetros, os três parâmetros foram escolhidos de acordo com o trabalho de (LI et al., 2014a) que classifica o número de *mappers*, o número de *reducers* e a taxa de *slowstart* como pertencentes a mesma categoria e que seus valores ótimos para esses parâmetros são específicos da aplicação. Os parâmetros que foram escolhidos (número de *mappers*, número de *reducers* e valor de *slow start*) são descritos na Tabela 2. Esses parâmetros podem ser classificados da seguinte maneira, de acordo com as classificações já citadas na Seção 2.2.3:

- Em (LI et al., 2014), os parâmetros são pertencentes à *Categoria B*.
- Em (WU; GOKHALE, 2013), os parâmetros usados são classificados como *Parâmetros relevantes para a Hadoop*.
- Em (LI et al., 2014a), os parâmetros do Hadoop utilizados também podem ser classificados como pertencentes ao *Grupo 1*.

Tabela 2 – Parâmetros utilizados na abordagem.

Parâmetro	Descrição	Valor *default*
mapreduce.input.fileinputformat.split.minsize	O tamanho mínimo que parte da entrada que deve ser dividida	0
mapreduce.input.fileinputformat.split.maxsize	O tamanho máximo que parte da entrada que deve ser dividida	1GB
mapreduce.job.maps	O número padrão de tarefas maps por trabalho	2
mapreduce.job.reduces	O número padrão de tarefas reduces por trabalho	1
mapreduce.job.reduce.slowstart.completedmaps	Fração do número de maps na tarefa que devem ser concluída antes que os reduces iniciem os trabalhos	0.05

Fonte: Elaborada pelo autor. (2015)

4.3 Implementação

Um protótipo de pesquisa foi implementado baseado na arquitetura da ANICAH descrita na Seção 4.2. O objetivo do protótipo é permitir validar a abordagem proposta através de experimentos reais. O protótipo foi desenvolvido em Java e *Bash script* para a versão 2.2.0 do Hadoop, está disponibilizado sob uma licença livre e está disponível na plataforma Gitlab [12]. As técnicas de aprendizado de máquina são implementadas pela biblioteca Weka (WEKA, 2011). Atualmente, o protótipo tem suporte à infraestrutura de computação em nuvem OpenStack, mas pode ser facilmente adequado a outras infraestruturas de computação em nuvem privadas e públicas como a IBM Cloud, a Amazon ou Azure. Os princípios essenciais do projeto do protótipo são a transparência e a autonomia. Ou seja, a sua utilização não implica em alteração de códigos das aplicações e nem do Hadoop. Além disso, o protótipo possui um comportamento autônomo ao buscar automaticamente por configurações do Hadoop que permitam reduzir o tempo de execução das aplicações.

A utilização do protótipo é simples e automatizada. Para utilizar o protótipo, é necessário apenas executar o seu binário em vez de executar o binário do Hadoop. O protótipo se encarrega de realizar o treinamento e iniciar a aplicação baseada na configuração que alcançou o melhor desempenho da aplicação. A Figura 25 mostra o funcionamento do protótipo como descritos nos seguintes passos:

1. Treinamento. É realizado através da entrada da aplicação MapReduce escrita em Java (.jar) e da entrada. É executado um código *Bash Script* que irá escrever em um arquivo de saída as configurações e seus respectivos tempos de execução.

[12] https://gitlab.com/

2. Geração do modelo preditivo. Em suma, a análise preditiva usa dados, algoritmos estatísticos e técnicas de Aprendizagem de máquina para identificar a probabilidade de resultados futuros com base em dados históricos[13]. Então, o modelo preditivo é um tipo de aprendizagem supervisionada que pode utilizar classificação ou regressão, nesse trabalho utilizou-se os modelos regressivos. Eles são criados através das técnicas de Aprendizagem de Máquina que o usuário optar. Atualmente, estão disponíveis as seguintes abordagens: Redes de Funções de Base Radial, Regressão Linear, Máquinas de Vetores de Suporte, Processos Gaussianos e Perceptron de Múltiplas Camadas, que são implementadas pela biblioteca Weka (WEKA, 2011). A implementação atual utiliza cada técnica de maneira modular e o usuário deve escolher qual delas usar. Para fins de avaliação, foram utilizadas as cinco técnicas, uma por vez, no intuito de observar como elas se comportam diante de dadas aplicações (nesse caso, WordCount e Terasort que serão vistas no Capítulo 5). São estabelecidos os parâmetros de configuração (se forem necessários) e gerados modelos de regressão (`.model`) a partir das instâncias de treinamento que retornará um valor real (`double`) a partir de uma configuração do tipo (número de *mappers*, número de *reducers*, valor de *slowstart*) especificada de AG.

3. Execução do Algoritmo Genético. A implementação do Algoritmo Genético para este problema foi a seguinte. O cromossomo ou indivíduo para o AG é uma configuração do tipo (*mappers*, *reducers*, *slowstart*), isto é, o que será avaliado na população é quão boa uma configuração se apresenta para melhorar o desempenho da aplicação. O cromossomo é composto por 3 valores (genes) do tipo *double*.

   Ao iniciar a execução do AG, há uma geração aleatória de cada indivíduo (pertencente a uma geração) que será avaliado. Na avaliação, o código recebe o caminho do arquivo de treinamento onde os registros são do tipo (*mappers*, *reducers*, *slowstart*, *tempo*), transformando-os em instâncias de dados. Depois disso, é criado o *model* com a técnica de aprendizagem de máquina escolhida recebendo com parâmetro as instâncias de dados do treinamento. Após isso, para a previsão, o modelo recebe como instância a configuração que deseja ter o tempo de execução previsto dessa forma (*mappers*, *reducers*, *slowstart*, ?), onde a interrogação representa o tempo que ainda não é conhecido, gerando o tempo desconhecido e atribuindo à variável de aptidão. Após isso, o indivíduo é adicionado à população, até que esta população inicial chegue ao tamanho especificado (para essa implementação foi 10).

   Após isso, o código do AG recebe o número de gerações que serão criadas (nesse caso 100) e o processo que será descrito abaixo irá se repetir essa quantidade de vezes. Tendo uma estrutura de repetição até o tamanho de cada geração (nesse caso 10), escolhe-se um índice aleatório até o tamanho da geração (*randomIndex*) e outro

[13] http://www.sas.com/pt_br/insights/analytics/analise-preditiva.html

índice randômico de 0 a 2 que diz respeito a posição do gene (*partOfChromossome*). Então, o cromossomo da posição da estrutura de repetição (ou seja, de 0 a 10, nesse caso) e o cromossomo da posição *randomIndex* serão cruzados. Nessa etapa serão feitos dois cruzamentos que são invocados por um cromossomo e passa como parâmetro o outro cromossomo que é especificado pelo *randomIndex* e o gene que será modificado. Após isso, esses dois cromossomos gerados no cruzamento passarão pela mutação.

Na mutação, a variável *turnOnMutation* vai receber um valor gerado aleatoriamente entre 0 e 4. Caso o valor seja diferente de 0, a mutação é ativada. Quando ativada, gera-se aleatoriamente um valor entre 0 e 2 que diz respeito ao gene que será mutado. Depois disso, o cromossomo é avaliado, isto é, o tempo de execução previsto é atribuído à aptidão do cromossomo. Após isso, esses dois cromossomos cruzados e, talvez mutados, são adicionados à população.

Após isso a população será atualizada para próxima iteração (de 1 a 100). Nessa etapa os indivíduos mais aptos da geração atual são selecionados e o indivíduo tem uma probabilidade de ser selecionado proporcional à sua aptidão. Nesse código, a população é ordenada e cria-se um *ArrayList* de cromossomos para guardar a nova população. Nesse caso, se a população inicial tem 10 indivíduos e são gerados 2 indivíduos por vez através de dois cruzamentos sendo realizado 10 vezes, existirá então uma população de 30 indivíduos a serem selecionados para nova população. Com a ordenação dos cromossomos, as três primeiras posições possuem as 3 melhores configurações que são diretamente armazenadas na nova população.

Os remanescentes são selecionados pelo método da roleta. Para visualizar este método considere um círculo dividido em n regiões (tamanho da população), onde a área de cada região é proporcional à aptidão do indivíduo. Coloca-se sobre este círculo uma "roleta"com n cursores, igualmente espaçados. Após um giro da roleta a posição dos cursores indica os indivíduos selecionados. Este método é denominado amostragem universal estocástica[14]. Nesse caso, gera-se um valor aleatório chamado *roullete* de 0 a 29. Se esse valor for:

Menor que 15 gera-se um valor aleatório de 0 a 10 que é somado a 4 e indica o índice do cromossomo que será adicionado a população. Ou seja, escolhe um cromossomo entre os indivíduos 4 e 10.

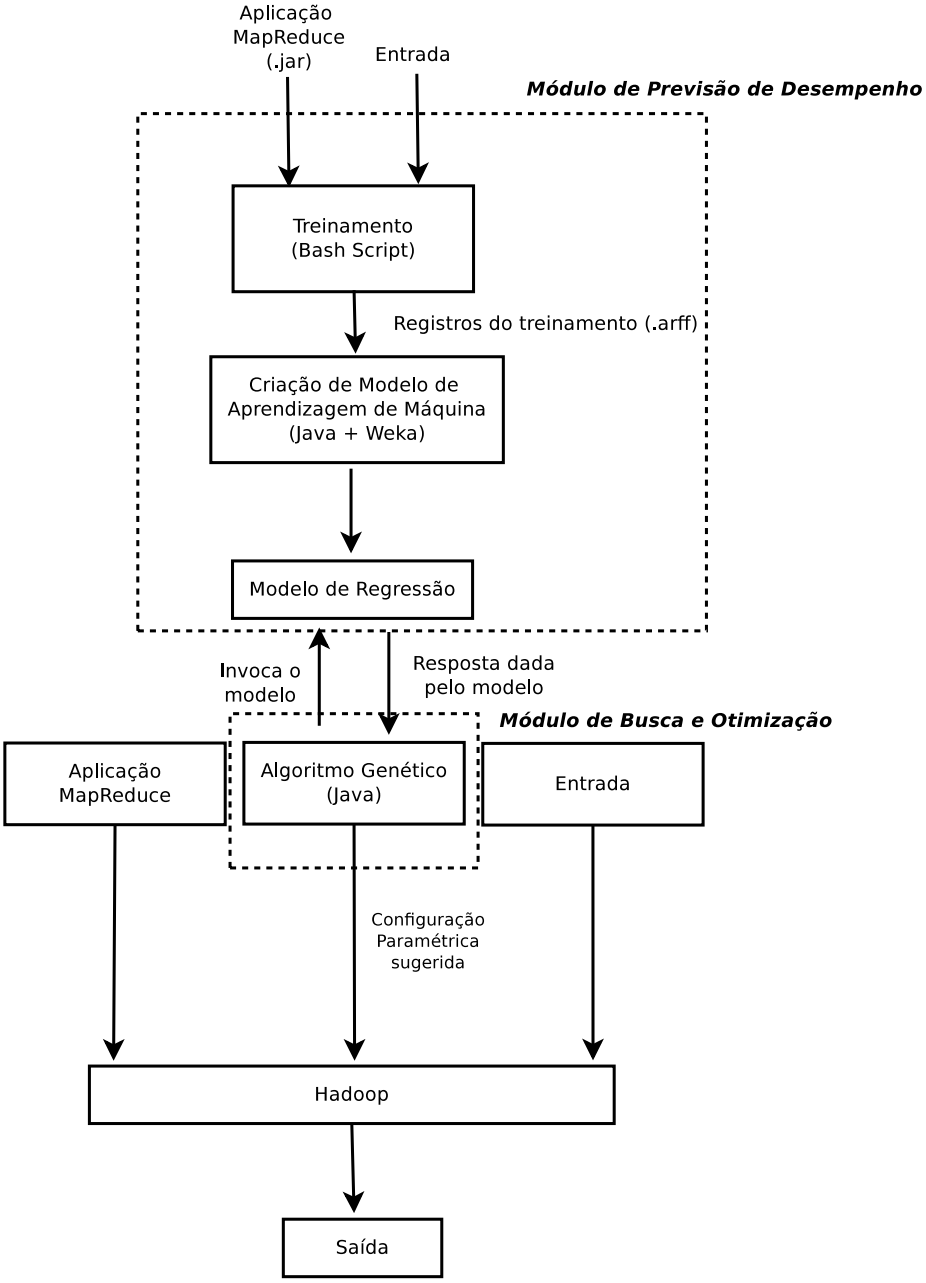
Maior ou igual a 15 e menor que 27 gera-se um valor aleatório de 0 a 15 que é somado a 11 e indica o índice do cromossomo que será adicionado a população. Ou seja, escolhe um cromossomo entre os indivíduos 11 e 25.

[14] http://www.nce.ufrj.br/GINAPE/VIDA/alggenet.htm

Maior ou igual a 27 e menor ou igual a 30 gera-se um valor aleatório de 0 a 5 que é somado a 26 e indica o índice do cromossomo que será adicionado a população. Ou seja, escolhe um cromossomo entre os indivíduos 26 e 30.

Esse processo repete-se até completar o tamanho 10 da nova população. Desse modo, o melhor cromossomo da última geração, está na posição 0 do ArrayList relativo à população final, corresponde à configuração sugerida pela ANICAH.

Figura 25 – Funcionamento geral da implementação do protótipo da solução.



Fonte: Elaborada pelo autor. (2015)

5 AVALIAÇÃO

Essa Seção diz respeito à avaliação da abordagem através do protótipo implementado para atestar e validar a eficiência e eficácia da solução. O uso dos Modelos preditivos no Algoritmo Genético, descritos em 2.3, visa conseguir o equilíbrio entre o desempenho (através da previsão de tempo de execução) e precisão (através da escolha de valores mínimos pelo AG). Para observar o comportamento da abordagem implementada, dois estudos de casos foram escolhidos e são descritos a seguir, sendo eles pertencentes ao conjunto de aplicações oficiais de teste do Hadoop.

Esse Capítulo está organizado da seguinte maneira: a Seção 5.1 refere-se ao Treinamento para os estudos de caso. A Seção 5.2 descreve os dois estudos de caso (*Wordcount* e *Terasort*). A Seção 5.3 descreve o cenário construído para as execuções dos testes e, por fim, a Seção 5.4 descreve os resultados alcançados.

5.1 Treinamento

Na criação do conjunto de treinamento foi medido o tempo de execução das aplicações Hadoop relativo a combinações dos parâmetros utilizados pela a ANICAH (ver Tabela 2, Seção 4.2). Isto é, o registro dos tempos de execução obtidos por meio da execução real por linha de comando de uma dada aplicação MapReduce com o Hadoop. Os tempos são registrados juntamente com a configuração utilizada para a obtenção do tempo de execução respectivo (número de *mappers*, número de *reducers* e valor de *slow start*) como mostrado no Apêndice **??** e Apêncice **??**. Em relação ao número de *mappers* no estudo de caso Wordcount são utilizados os parâmetros relacionados ao tamanho do *split* para configurar a quantidade correta de *mappers*. Os dois parâmetros relativos ao *split size* irão garantir o número de *mappers*, por isso serão considerados um único parâmetro. Já no estudo de caso Terasort é usado apenas o parâmetro *mapreduce.job.maps* para determinar o número de *mappers*.

Em um segundo momento, com alguns valores obtidos através das execuções, criou-se um conjunto de treinamento, montado automaticamente e variando os valores dos parâmetros, que é primordial para a criação dos modelos de regressão em Java através da biblioteca Weka (WEKA, 2011) que é uma coleção de algoritmos de aprendizagem de máquina para tarefas de mineração de dados. A seguir, temos alguns parâmetros importantes utilizados para cada uma das técnicas de aprendizagem de máquina implementadas por meio do Weka para a predição do tempo de execução para uma respectiva configuração.

- Redes de Funções de Base Radial (RBF): semente de agrupamento=8, fator de crista=0.7 e desvio padrão mínimo para os clusters=0.1.

- Regressão Linear (LR): método de seleção de atributo para ser usado=1.
- Máquinas de Vetores de Suporte (SVM): tolerância para verificar o critério de parada=0.01.
- Processos Gaussianos (GP): parametrização *default*.
- Perceptron de Múltiplas Camadas (MLP): parametrização *default*.

A implementação da etapa de treinamento foi realizada em *Bash script*, invocando a execução do Hadoop diretamente no terminal (interpretador de comandos Bash). O conjunto de treinamento é composto de 125 configurações, mostrada na segunda coluna da Tabela 3, e seus respectivos tempos de execução onde os valores assumidos encontram-se nos Apêndices **??** e **??** desse estudo. Com esse conjunto de treinamento, são criados os modelos de regressão para prever o tempo de execução dos valores não conhecidos dentro do Algoritmo Genético, como mostrado na Seção 4.3.

Tendo os modelos gerados pelas técnicas de aprendizado de máquinas utilizadas para estimar o desempenho, implementou-se o Algoritmo Genético (como descrito na Seção 4.2) de modo que foram especificadas o número de gerações sendo igual a 100 e o tamanho da geração sendo 10 indivíduos. Além disso, é válido citar que o protótipo foi configurado para utilizar todas as técnicas de predição citadas para fins de observação das abordagens implementadas. O Algoritmo Genético que varia como mostrado na terceira coluna da Tabela 3, isto é, o Algoritmo Genético pode assumir 320 configurações.

Tabela 3 – Espaço de busca alcançado pelo treinamento e pelo Algoritmo Genético.

Parâmetro	Espaço de busca do Treinamento	Espaço de busca do Algoritmo Genético
número de *mappers*	1,2,4,6,8	1:8::1
número de *reducers*	1,2,4,6,8	1:8::1
valor de *slow start*	0.05,0.2,0.4,0.6,0.8	0.05,0.2,0.4,0.6,0.8

Fonte: Elaborada pelo autor. (2015)

Métricas de avaliação

O objetivo das métricas de avaliação é identificar quais são as técnicas de aprendizado de máquina mais adequadas, dada uma aplicação, sua entrada e o ambiente de execução. Desse modo, é válido lembrar que uma técnica boa em determinado contexto pode ser ruim em outro. Devido a isso, elas se apresentam de maneira modular, podendo ser substituídas facilmente por outros modelos de predição.

As métricas empregadas para avaliar a abordagem proposta são *eficiência* e *eficácia* (LIAO; DATTA; WILLKE, 2013). A qualidade do tempo de busca é expressa pela métrica *eficiência*. Nesse caso, considere tempo de busca como sendo o tempo que o AG leva para encontrar uma configuração, sem considerar o tempo de treinamento que é o

mesmo para as cinco técnicas. Ou seja, dadas algumas abordagens para uma mesma tarefa, aquelas que entregarem o resultado ou resposta mais rapidamente são consideradas mais eficientes. Para tal fim, é medido o tempo de busca que uma técnica leva para entregar uma solução. Por outro lado, a *eficácia* permite mensurar a qualidade das configurações encontradas pelo protótipo, ou seja, permite mensurar o tempo de execução da aplicação Hadoop. Dessa maneira, a busca da configuração deve ser eficiente, ela deve ser alcançada em um tempo de busca razoável. Ao mesmo tempo, essa configuração deve resultar em um melhor desempenho da aplicação (ser eficaz), isto é, no mínimo, um tempo de execução menor do que a configuração padrão do Hadoop.

5.2 Estudos de caso

Wordcount

A implementação do algoritmo WordCount é proposta para fins de teste e no intuito de demonstrar o funcionamento básico do MapReduce. O WordCount lê arquivos de texto e realiza a contagem de quantas vezes as palavras aparecem. A entrada consiste em arquivos de texto e a saída são arquivos de texto onde cada linha dos quais contém uma palavra e a contagem de quantas vezes ela ocorre, separadas por um guia. Cada *mapper* adota uma linha como entrada e quebra-a em palavras. Em seguida, ele emite um par chave/valor da palavra e 1. Cada *reducer* soma as contagens para cada palavra e emite uma única chave/valor com a palavra e soma. Como uma otimização, o *reducer* é também utilizado como um combinador nas saídas *mapper*. Isso reduz a quantidade de dados enviados pela rede, combinando cada palavra em um único registro (HADOOP, 2008). O pseudocódigo mostrado anteriormente na Figura 5 (ver Seção 2.2.1), descreve como a aplicação lida com a entrada, gera os dados intermediários e obtém a saída.

De maneira detalhada, um arquivo de texto qualquer é armazenado no HDFS. Em um primeiro momento, a entrada chave/valor do algoritmo é o número da linha e o conteúdo da linha em questão. Essa é a entrada para a primeira função do MapReduce, a função *Map*. Nesta função o objetivo é processar a linha do arquivo para extrair todas as palavras do texto e, para cada palavra, adicionar um contador com o valor 1. Quando a função *Map* é finalizada, um passo intermediário é executado na intenção de ordenar lexicograficamente as chaves emitidas na função *Map*. A função *Reduce* é invocada com o objetivo de contar quantas vezes uma determinada chave apareceu, para cada ocorrência da chave ela irá incrementar o seu valor em 1. A saída será todas as ocorrências de palavras distintas encontrada no texto e a quantidade de vezes que ela apareceu. A Figura 7 (ver Seção 2.2.2) mostra um exemplo dessas etapas.

Terasort

O Terasort é um pacote composto por 3 aplicações MapReduce para Hadoop (HADOOP, 2008): *TeraGen*, *Terasort* e *TeraValidate*. O *Teragen* é um programa Hadoop para gerar os dados. Divide-se o número desejado de linhas pelo número desejado de tarefas e atribui-se intervalos de linhas para cada *mapper*. O *mapper* salta o gerador de números aleatórios para o valor correto para a primeira linha e gera as linhas seguintes.

O *TeraSort* é um padrão *sort* MapReduce, com exceção de um particionador personalizado que usa uma lista ordenada de $N-1$ chaves amostradas que definem a faixa de chaves para cada *reducer*. Em particular, todas as chaves encontram-se de tal forma que amostra $[i-1] <= chave < amostra[i]$ são enviados para *reducer* i. Isso garante que a saída do *reducer* i é menor do que a saída do *reducer* $i+1$.

O *TeraValidate* assegura que a saída está globalmente classificada. Ele cria um *mapper* por arquivo no diretório de saída e cada *mapper* garante que cada chave é inferior ou igual ao anterior. O *mapper* também gera registros com as primeiras e últimas chaves do arquivo e a função *reduce* garante que a primeira chave de arquivo i é maior que a última chave de arquivo $i-1$. Todos os problemas são relatados como saída *reducer* com as chaves que estão fora de ordem.

5.3 Cenários

Configuração Experimental

Experimentos foram realizados utilizando a infraestrutura de nuvem OpenStack (OPENSTACK, 2015) do Laboratório de Computação Científica e Visualização (LCCV)[15]. O protótipo utilizou o Hadoop 2.2.0 em um cluster com máquinas virtuais do tipo *small*, que utiliza uma VCUP (*virtual CPU*), com memória principal de 2 GB. Ao todo, 4 máquinas virtuais foram utilizadas, sendo um o nó mestre (*master*) e as três demais os trabalhadores (*workers*). Os resultados foram comparados entre si e com a configuração *default*.

Para essa configuração foi realizada a execução da abordagem para cada uma das cinco técnicas de aprendizagem de máquina utilizadas para a predição de desempenho; e para cada estudo de caso. Desse modo, foi realizado o treinamento de 125 configurações, a criação dos modelos de regressão que são utilizados no Algoritmo Genético que, por fim, irá retornar uma configuração de parâmetros melhor que a parametrização padrão do Hadoop. A entrada utilizada foi de 1GB. Para fins de verificação dos tempos de execução nos testes foi realizado o registro desses tempos relativos às 320 configurações possíveis para o Algoritmo Genético.

[15] http://www.lccv.ufal.br/

5.4 Resultados

Os experimentos foram realizados com a implementação da arquitetura ANICAH para propor automaticamente a configuração das aplicações Wordcount e Terasort, de acordo com a configuração especificada na Seção 5.3. Isto é, as configurações sugeridas foram executadas na mesma estrutura e o tempo real calculado. Foram obtidas as configurações do Hadoop presentes na Tabela 4 e Tabela 5, respectivamente, que são descritos a seguir.

- Para o Wordcount, o protótipo sugeriu a configuração de 8 *mappers* enquanto a configuração do número de *reducers* variou entre 1, 7 e 8. O valor do parâmetro *slow start* assumiu os valores de 0.05 ou 0.8.

- Já para o Terasort, foi sugerida a configuração de 2, 6 e 8 *mappers* enquanto a configuração do número de *reducers* variou entre 1 e 2. O valor do parâmetro *slow start* variou entre os valores 0.05 e 0.2.

Primeiramente, observou-se que para as 10 configurações sugeridas (mostrada nas Tabelas 4 e 5), 7 configurações sugerem para o valor do parâmetro *slow start* o valor 0.05, que como podemos ver na coluna Default das tabelas é o valor padrão estabelecido pelo próprio Hadoop. Desse modo, acredita-se que tal valor foi fixado por conseguir abranger uma gama de aplicações fornecendo um bom desempenho.

Tabela 4 – Valores dos parâmetros encontrados para o Wordcount pela ANICAH para cada uma das técnicas que podem ser usadas para medir a taxa de aptidão no Algoritmo Genético.

Parâmetro	RBF(bytes)	LR(bytes)	SVM(bytes)	GP(bytes)	MLP(bytes)	Default
mappers	8	8	8	8	8	2
reduces	8	7	1	7	8	1
slow start	0.05	0.05	0.8	0.05	0.05	0.05

Fonte: Elaborada pelo autor. (2015)

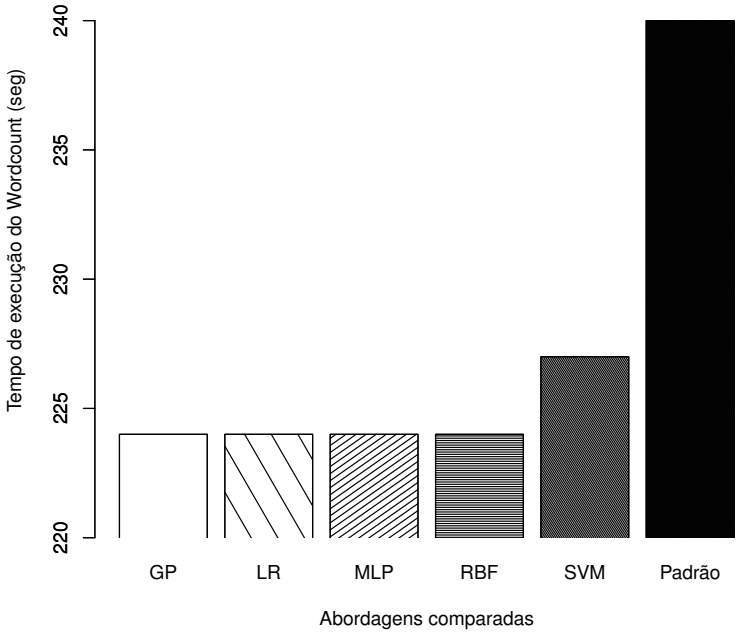
Tabela 5 – Valores dos parâmetros encontrados para o Terasort pela ANICAH para cada uma das técnicas que podem ser usadas para medir a taxa de aptidão no Algoritmo Genético.

Parâmetro	RBF(bytes)	LR(bytes)	SVM(bytes)	GP(bytes)	MLP(bytes)	Default
mappers	6	2	8	8	2	2
reduces	1	2	2	2	2	1
slow start	0.05	0.2	0.05	0.05	0.2	0.05

Fonte: Elaborada pelo autor. (2015)

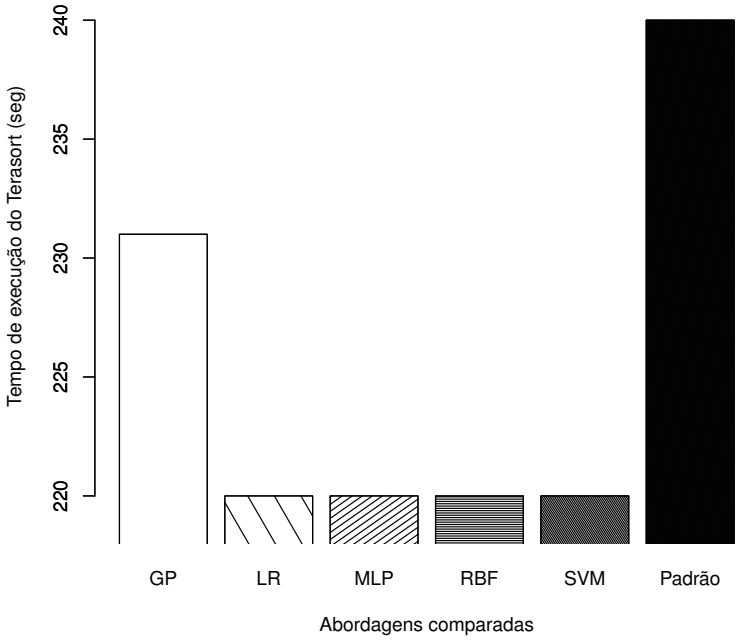
De acordo com os dados obtidos comparados à parametrização *default*, a ANICAH reduziu consideravelmente o tempo de execução da aplicação Hadoop, como mostrado na Figura 26 e na Figura 27. Ou seja, a ANICAH foi mais *eficaz* do que a configuração padrão do Hadoop (*default*). Com relação à *eficiência*, isto é, o tempo que os modelos preditivos gerados pelas técnicas de aprendizado de máquina combinadas com o Algoritmo

Figura 26 – Comparação do tempo de execução (*eficácia*) relativo à configuração recomendada entre as abordagens no Wordcount.



Fonte: Elaborada pelo autor. (2015)

Figura 27 – Comparação do tempo de execução (*eficácia*) relativo à configuração recomendada entre as abordagens no Terasort.

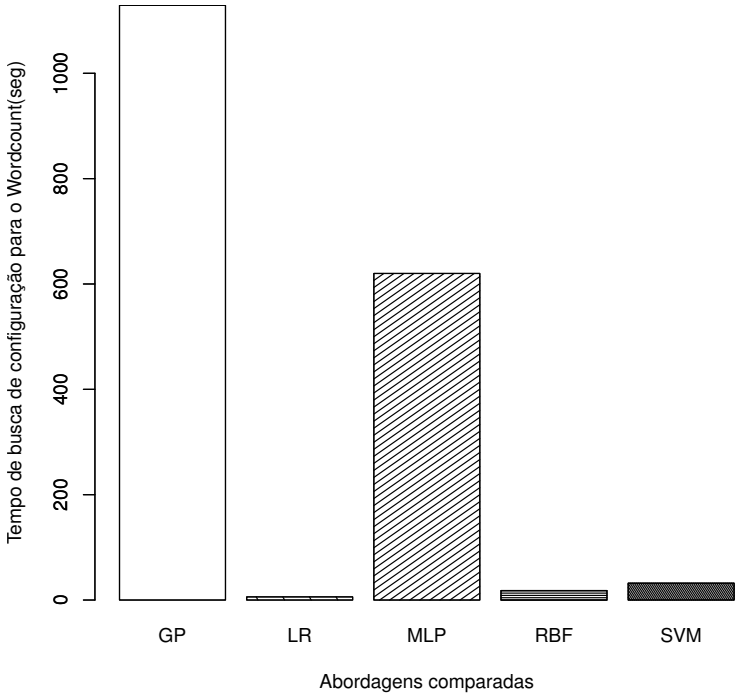


Fonte: Elaborada pelo autor. (2015)

Genético levaram para encontrar uma configuração para sugerir, foram encontrados resultados distintos. Para esses estudos de casos e seus contextos de execução, em ordem crescente de tempo de busca, tem-se: Regressão Linear (LR), Redes de Funções de Base Radial (RBF), Máquinas de Vetores de Suporte (SVM), Perceptron de Múltiplas Camadas (MLP), Processos Gaussianos (GP). Dentre elas, o Processos Gaussianos (GP) destaca-

se negativamente pois alcançou o maior tempo de busca. Dessa maneira, os resultados mostram que para o Wordcount e Terasort, a técnica de Processos Gaussianos (GP) consegue o pior equilíbrio entre eficiência e eficácia. Enquanto os melhores resultados para as mesmas métricas pertencem a Regressão Linear (LR) e Redes de Funções de Base Radial (RBF). É conveniente lembrar que as técnicas se encontram de maneira praticamente pura, e algumas parametrizações podem melhorar os resultados. Devido a ausência dessa parametrização, pela variação de aplicações Hadoop e contextos de execução, a qualidade da predição não pôde ser avaliada globalmente nesse trabalho.

Figura 28 – Comparação do tempo de busca da configuração (*eficiência*) para o Wordcount entre Redes de Funções de Base Radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP) e Perceptron de Múltiplas Camadas (MLP).

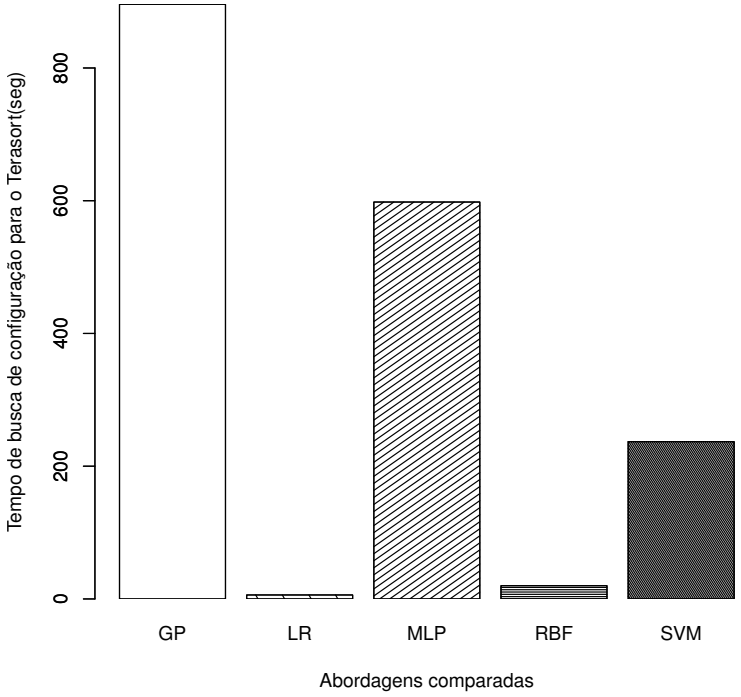


Fonte: Elaborada pelo autor. (2015)

Foram executadas as 320 configurações possíveis para o Algoritmo Genético com todas as combinações dentro do universo de busca proposto para saber como a ANICAH se comportou em relação à melhor opção dentre esse espaço de busca, visto nas Figura 30 e Figura 31. A abordagem encontra tempos de execução menores e a parametrização padrão do Hadoop, porém nem sempre encontra a solução ótima para o espaço de busca e o contexto em questão.

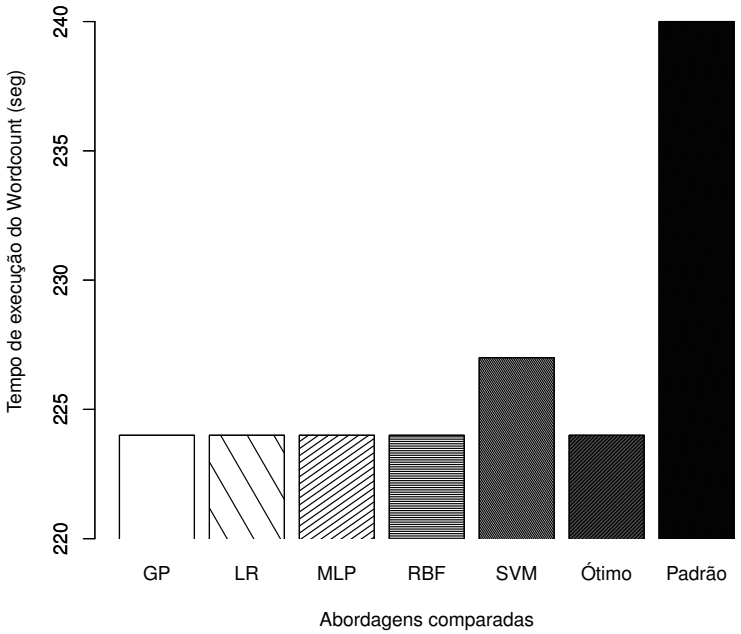
Por fim, o Algoritmo Genético utilizando os Modelos preditivos desenvolvido mostra-se mais rápido que a versão sem predição desse mesmo algoritmo. Tendo em vista que na validação do protótipo desenvolvido são criadas 100 gerações onde cada uma delas possui 10 indivíduos, o Algoritmo Genético realizará 1000 verificações de taxas de aptidão. No caso do AG sem predição, as configurações vão sendo otimizadas e assim o tempo de

Figura 29 – Comparação do tempo de busca da configuração (*eficiência*) para o Terasort entre Redes de Funções de Base Radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP) e Perceptron de Múltiplas Camadas (MLP).



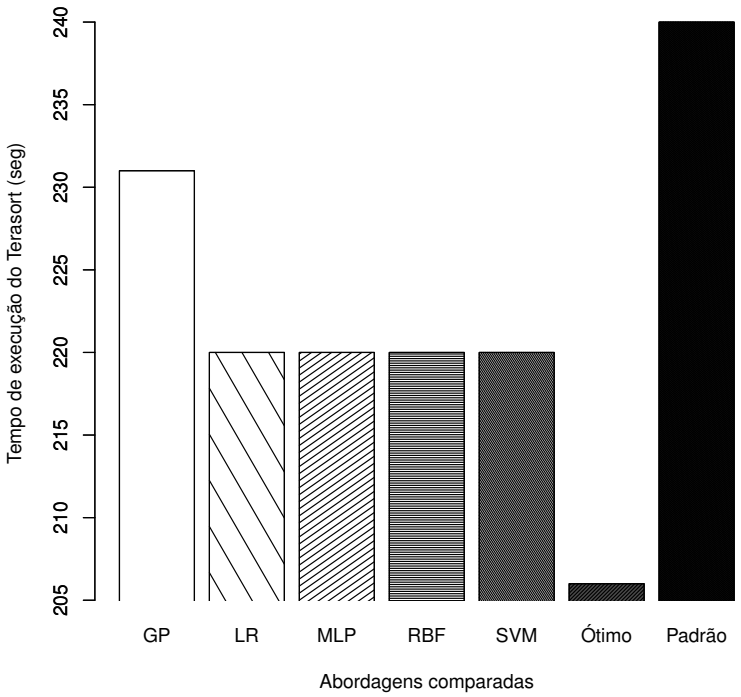
Fonte: Elaborada pelo autor. (2015)

Figura 30 – Comparação do tempo de busca da configuração (*eficiência*) para o Wordcount entre Redes de Funções de Base Radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP), Perceptron de Múltiplas Camadas (MLP) e a opção ótima para esse espaço de busca.



Fonte: Elaborada pelo autor. (2015)

Figura 31 – Comparação do tempo de busca da configuração (*eficiência*) para o Terasort entre Redes de Funções de Base Radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP), Perceptron de Múltiplas Camadas (MLP) e a opção ótima para esse espaço de busca.



Fonte: Elaborada pelo autor. (2015)

execução da aplicação atribuído à taxa de aptidão vai ficando menor para a verificação de cada individuo. Porém como cada taxa de aptidão é relativa a uma execução no Hadoop, isso leva um tempo razoável (para esses estudos de casos mais de 200 segundos). No caso do AG com modelos preditivos, são realizadas as 125 execuções pertencentes ao conjunto de treinamento, mas depois disso não serão necessárias as execuções no Hadoop durante o funcionamento do AG. Para encontrar a taxa de aptidão serão feitas apenas as invocações do modelo de predição que são bem mais rápidas que os tempo de execução encontrados para as configurações pertencentes ao espaço de busca do AG (para esses estudos de casos de modo específico, o LR e RBF conseguem tornar o AG Modificado mais eficiente). Desse modo, esse AG com Modelos Preditivos levará menos tempo para encontrar uma configuração que o AG sem predição.

5.5 Limitações metodológicas

Esse trabalho possuiu alguns fatores limitantes para metodologia que foram o tempo e a infraestrutura disponível. Primeiramente, foram escolhidos apenas os três parâmetros citados anteriormente (número de *mappers*, número de *reducers* e valor de *slowstart*) para serem manipulados. Grande parte dos trabalhos relacionados utiliza um número elevado de parâmetros manipulados, chegando a mais de 20 parâmetros configurados simultane-

amente. Porém, pela limitação de tempo, não se pôde fazer um estudo exploratório dos mais de 200 parâmetros para atestar e utilizar aqueles que obtiverem, comprovadamente, maiores impactos sobre o desempenho das aplicações. Desse modo, estabeleceu-se poucos parâmetros para reduzir o espaço de busca, e consequentemente os tempos de busca, selecionando-os pela maior utilização no estado da arte e pela própria definição da sua função no ciclo de vida de uma aplicação Hadoop.

Outro ponto que teve que ser limitado pelo tempo e infraestrutura foi a determinação dos espaços de busca do treinamento e do Algoritmo Genético para o protótipo de avaliação. Em primeiro lugar, o espaço de busca do AG (ou da própria ANICAH) foi limitado a 320 possibilidades com: 8 valores para *mappers*, 8 valores para *reducers* e 5 valores para *slowstart*. Isso aconteceu devido a infraestrutura que se tinha disponível para realização dos testes, de modo que as configurações possíveis pudessem sem executadas adequadamente na infraestrutura de nuvem disponível. Tais valores possíveis para cada parâmetro (visto na Tabela 3) foram estabelecidos para conseguir distribuir de maneira uniforme os pontos na curva que o Algoritmo Genético vai investigar.

Em relação ao espaço de busca do treinamento, 125 possibilidades foram estabelecidas para também espaçar de maneira uniforme os pontos (ou configurações) para conseguir criar modelos preditivos de maior precisão. Um vez que a aplicação pode se comportar de maneira linear ou não, um maior número de pontos consegue criar modelos mais condizentes com o comportamento real da aplicação. Para aplicações de comportamento linear poucos pontos poderiam criar modelos bons, porém para aplicações de comportamento não linear um pequeno número de pontos pode criar modelos que comprometam a acurácia da previsão. Por isso, escolheu-se uma quantidade razoável de pontos, considerando o espaço de busca da ANICAH, que não prejudicasse em relação ao tempo e conseguisse abranger a criação de modelos com tendências lineares e não lineares.

Em segundo lugar, a etapa que registra os tempos de execução das aplicações Hadoop com diversas configurações foi executada apenas uma vez. Isso aconteceu devido ao tempo que se tinha disponível para realização dos testes e finalização do trabalho, e pela infraestrutura de nuvem que permaneceu *offline* durante um tempo considerável. Desse modo, é recomendável reexecutar algumas vezes a criação do conjunto de treinamento para observação da variância dos tempos de execução e poder gerar resultados mais precisos e confiáveis. Além disso, também pela ausência de tempo, não se pode determinar através de várias execuções da ANICAH a porcentagem de acertos da abordagem (isto é, resultando em configurações com menores tempos de execução) comparados com escolhas de configurações aleatórias, por exemplo.

Por fim, o tempo limitou investigações relacionadas aos dados de entrada em relação ao tempo de execução. Primeiramente, não se pôde atestar se entradas distintas podem impactar em diferentes tempos de execução da mesma aplicação Hadoop. Além disso, também não foi possível ser estudada qual é a proporção do impacto do tamanho dos

dados de entrada. Ou seja, tendo apenas a execução de uma parcela da entrada, como inferir seu comportamento proporcional ao crescimento dos dados. Isso deve ser feito assim que possível para entender os impactos da entrada para a ANICAH.

6 CONCLUSÕES

Nos últimos anos, a quantidade de dados gerados no mundo tem aumentado de maneira significativa. Nesse contexto, a Computação em Nuvem juntamente com o Modelo de Programação MapReduce, através do arcabouço Hadoop, têm sido utilizados para o processamento dessa grande quantidade de dados. Contudo, os sistemas contemporâneos ainda são complexos e dinâmicos, podendo mudar seu contexto e suas necessidades, e tornando-se cada vez difíceis de se configurar. De uma maneira mais específica, configurar aplicações Hadoop é uma tarefa complexa devido à quantidade e à complexidade dos seus parâmetros. Desse modo, a configuração automática desses parâmetros para aplicações Hadoop é uma alternativa para diminuir a complexidade de programação e adaptar essas aplicações a ambientes dinâmicos.

Esse trabalho propõe uma solução automática para a configuração de parâmetros do arcabouço MapReduce Hadoop com o objetivo de diminuir o tempo de execução das aplicações. São manipulados alguns dos principais parâmetros de configuração do Hadoop, sendo eles o número de *mappers*, o número de *reducers* e o valor de *slow start*. A abordagem proposta utiliza um Algoritmo Genético cuja função de aptidão é mensurada através de técnicas de predição (aprendizagem de máquina) que são treinadas previamente. Portanto, é mostrada uma arquitetura modular que aceita qualquer algoritmo de Aprendizagem de Máquina que gere modelos de regressão, e isso foi demostrado com os modelos das técnicas: Redes de Funções de Base Radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP) e Perceptron de Múltiplas Camadas (MLP). Além disso, a solução proposta é totalmente transparente, ou seja, não requer modificação nem da aplicação MapReduce nem do Hadoop.

A primeira conclusão desse trabalho é a constatação de que as técnicas de predição aplicadas no Algoritmo Genético utilizado permite melhorar o tempo de execução de aplicações Hadoop. Isso acontece pela habilidade do Algoritmo Genético encontrar, nesse caso, o menor valor de aptidão que é relativo ao tempo de execução de uma aplicação e também pela capacidade do Modelo preditivo conseguir prever o tempo de execução de uma aplicação sem executá-la realmente. A consequência maior desta conclusão é que programadores iniciantes ou experientes que utilizarem essa solução conseguem um tempo de execução reduzido para as aplicações, sem precisar conhecer profundamente os parâmetros do Hadoop e seus espaços de busca.

A contribuição desse trabalho se diferencia das demais abordagens para Configuração Automática de parâmetros do Hadoop principalmente por combinar o Algoritmo Genético com Modelos preditivos, no caso gerados por técnicas de aprendizagem de máquina. Isso acontece através da inserção do uso das técnicas para a previsão de desempenho baseadas em regressão no Algoritmo Genético. Elas irão inferir o tempo de execução da aplicação

Hadoop para uma configuração e esse valor é atribuído a taxa de aptidão. Desse modo, o Algoritmo Genético vai buscar o menor tempo de execução baseado na taxa de aptidão. Ao final de sua execução, nem todas as configurações investigadas pelo Algoritmo Genético precisaram necessariamente serem executadas pelo Hadoop.

A abordagem implementada possui algumas vantagens em relação às técnicas que utilizam Algoritmo Genético puro ou outras técnicas de aprendizado de máquina. Primeiramente, a abordagem proposta utiliza um menor número de parâmetros, reduzindo consideravelmente o espaço de busca e aumentando a possibilidade de encontrar mais facilmente uma melhor configuração. Em seguida, por se tratar de uma abordagem de configuração automática estática (antes de executar a aplicação), as curvas de aprendizados são mais rápidas proporcionando uma maior facilidade de reprodução da solução e manutenção do código. A abordagem desenvolvida também possui uma arquitetura modular, extensível e configurável, permitindo a utilização e a adição de diferentes técnicas de aprendizado de máquina para predição no Algoritmo Genético. Além disso, a arquitetura da ANICAH inova por ser transparente, independente e não intrusiva. Nesse caso, a configuração automática resolve o problema como uma caixa-preta, onde não se tem acesso ao código e portanto não se pode alterar internamente o código do Hadoop ou da aplicação MapReduce. Esse software está disponibilizado publicamente como software livre, juntamente com suas instruções para reprodução do trabalho.

6.1 Trabalhos Futuros

A abordagem proposta nesse trabalho abre várias possibilidades de pesquisa. Primeiramente, abordagem pode utilizar a Configuração Automática Dinâmica, uma vez que ela permite a adaptação da aplicação enquanto está sendo executada. Por exemplo, inspirando-se nos trabalhos de (LI et al., 2014a) e (LI et al., 2014). Entretanto, deve-se evitar o impacto negativo da configuração dinâmica no desempenho. Isso pode ser implementado da seguinte maneira: a criação de um módulo *Verificador* que analisa as informações de log das execuções da etapa de Treinamento realizada para os modelos de regressão da ANICAH. A partir dessas execuções, o *Verificador* entenderá quais as necessidades de variação de valores de parâmetros através de estratégias de análise de Relacionamento de fatores de impacto. A saída de tal módulo seria uma coleção de configurações candidatas que supram as diferentes necessidades ao longo do ciclo de vida do trabalho MapReduce, pois as características do trabalho e a utilização do cluster são dinâmicas e a aplicação deve se adaptar a essas variações. O segundo módulo seria o *Configurador Dinâmico* que vai conter o algoritmo de configuração, nesse caso o Algoritmo Genético Modificado desenvolvido, e as estratégias que a partir do que o *Verificador* resultar irá dicidir quais dos parâmetros devem ser modificados e que valores devem ser atribuídos. Desse modo, os dois módulos em conjunto devem analisar como as variações de configuração influenciam o desempenho e ajustar esses valores dinamicamente, isto é,

com a aplicação MapReduce já em execução.

Outra possibilidade de melhoria na abordagem proposta nesse trabalho é a inserção de outros parâmetros opcionais de configuração do Hadoop. Para isso, pode-se utilizar o trabalho de (WANG; LIN; TANG, 2012) que só ativa as modificações nos parâmetros que julga importantes para tal aplicação. Ao ativar a utilização de parâmetros, o espaço de busca dos parâmetros é ampliado. Por exemplo, ativando a utilização dos parâmetros *mapreduce.tasktracker.map.tasks.maximum*, que define o número de tarefas *map* que podem ser executadas simultaneamente, e *mapreduce.tasktracker.reduce.tasks.maximum*, que define o número de tarefas *reduce* que podem ser executadas simultaneamente. Assim, a utilização de cinco parâmetros, permitindo 8 valores para o número de *mappers*, número de máximo de tarefas *map*, número de *reducers* e número de máximo de tarefas *map*; e 5 valores para *slow start*, serão permitidas 20.480 configurações diferentes. Deve-se investigar se o tempo de busca das configurações ainda é viável, tendo em vista a considerável ampliação do espaço de busca assim como se a utilização desses ou outros parâmetros adicionais irão produzir configurações com tempos de execução menores que os encontrados com os parâmetros já avaliados no ANICAH (número de *mappers*, número de *reducers* e valor de *slow start*).

Esse trabalho pode ser utilizado também para o desenvolvimento de uma ferramenta de inovação tecnológica como um serviço na computação em nuvem na camada PaaS (plataforma como serviço). Esse serviço pode ser implementado baseado na arquitetura de Serviços Web RESTful (que são fracamente acoplados e interoperáveis), com modelo de negócios pagando por assinatura (*subscription*) ou por consumo (*pay-as-you-go*). Como a arquitetura ANICAH está em nível de PaaS, a implementação de tal serviço implica apenas em aspectos tecnológicos e de modelos de negócio.

Por fim, outro trabalho futuro consiste na implementação em um Módulo *Lembrar* que irá aprender com execuções antigas para cada aplicação MapReduce que for executada, como visto nos trabalhos futuros de (WU; GOKHALE, 2013). As abordagens de Configuração Automática de parâmetros (como o ANICAH) são recomendadas para serem utilizadas com grandes dados de entrada e quando se necessita utilizar várias vezes essa mesma aplicação MapReduce. Caso contrário, o esforço para encontrar melhores configurações quando se tem a intenção de usar poucas vezes a aplicação não será compensatório. Desse modo, a cada execução para uma aplicação deve ser armazenado seu log ou sua configuração e tempo de execução alcançado. Isso será feito no intuito de que após algumas execuções seja possível tomar decisões de escolha de configurações mais precisas a partir desses dados.

REFERÊNCIAS

BABU, S. Towards automatic optimization of mapreduce programs. In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. New York, NY, USA: ACM, 2010. (SoCC '10), p. 137–142. ISBN 978-1-4503-0036-0. Disponível em: <http://doi.acm.org/10.1145/1807128.1807150>.

BOUKERCHE, A. et al. Cloud-assisted computing for event-driven mobile services. *Mobile Networks and Applications*, Springer US, v. 19, n. 2, p. 161–170, 2014. ISSN 1383-469X. Disponível em: <http://dx.doi.org/10.1007/s11036-013-0488-1>.

Cisco Systems. *Visual Network Index (VNI): Forecast Highlights*. 2012. Http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html. Accessed in December 2013.

CORANA, A. et al. Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. Math. Softw.*, ACM, New York, NY, USA, v. 13, n. 3, p. 262–280, set. 1987. ISSN 0098-3500. Disponível em: <http://doi.acm.org/10.1145/29380.29864>.

COSTA, E. et al. A framework for building web mining applications in the world of blogs: A case study in product sentiment analysis. *Expert Systems with Applications*, v. 39, n. 5, p. 4813 – 4834, 2012. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417411014588>.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1327452.1327492>.

EBERHART, R.; SIMPSON, P.; DOBBINS, R. *Computational Intelligence PC Tools*. San Diego, CA, USA: Academic Press Professional, Inc., 1996. ISBN 0-12-228630-8.

FERNANDES, M.; A.D.D., N.; BEZERRA, J. Aplicação das redes RBF na deteção inteligente de sinais digitais. In: *IV Brazilian Conference on Neural Networks*. [S.l.: s.n.], 1999. v. 1, p. 226–230.

FILIERI, A.; HOFFMANN, H.; MAGGIO, M. Automated design of self-adaptive software with control-theoretical formal guarantees. In: *Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA: ACM, 2014. (ICSE 2014), p. 299–310. ISBN 978-1-4503-2756-5. Disponível em: <http://doi.acm.org/10.1145/2568225.2568272>.

FREITAS, A.; PARLAVANTZAS, N.; PAZAT, J.-L. An integrated approach for specifying and enforcing slas for cloud services. In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. [S.l.: s.n.], 2012. p. 376–383. ISSN 2159-6182.

HADOOP, A. *Hadoop*. 2008. Http://hadoop.apache.org/. Accessed in December 2013.

HILBERT, M.; LÓPEZ, P. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, American Association for the Advancement of Science, v. 332, n. 6025, p. 60–65, abr. 2011. ISSN 1095-9203. Disponível em: <http://dx.doi.org/10.1126/science.1200970>.

HORN, P. *Autonomic Computing: IBM's Perspective on the State of Information Technology*. [S.l.], 2001.

JADEJA, Y.; MODI, K. Cloud computing - concepts, architecture and challenges. In: *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*. [S.l.: s.n.], 2012. p. 877–880.

JOHANSSON, U.; LOFSTROM, T.; SONSTROD, C. Locally induced predictive models. In: *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. p. 1735–1740. ISSN 1062-922X.

KEPHART, J.; CHESS, D. The vision of autonomic computing. *Computer*, v. 36, n. 1, p. 41–50, Jan 2003. ISSN 0018-9162.

LAMA, P.; ZHOU, X. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In: *Proceedings of the 9th International Conference on Autonomic Computing*. New York, NY, USA: ACM, 2012. (ICAC '12), p. 63–72. ISBN 978-1-4503-1520-3. Disponível em: <http://doi.acm.org/10.1145/2371536.2371547>.

LI, C. et al. An adaptive auto-configuration tool for hadoop. In: *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*. [S.l.: s.n.], 2014. p. 69–72.

LI, H.; LI, Y.; LU, H. Semi-supervised learning with gaussian processes. In: *Pattern Recognition, 2008. CCPR '08. Chinese Conference on*. [S.l.: s.n.], 2008. p. 1–5.

LI, M. et al. Mronline: Mapreduce online performance tuning. In: *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*. New York, NY, USA: ACM, 2014. (HPDC '14), p. 165–176. ISBN 978-1-4503-2749-7. Disponível em: <http://doi.acm.org/10.1145/2600212.2600229>.

LI, X. et al. Price competition in a duopoly iaas cloud market. In: *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*. [S.l.: s.n.], 2014. p. 1–4.

LIAO, G.; DATTA, K.; WILLKE, T. L. Gunther: Search-based auto-tuning of mapreduce. In: *Proceedings of the 19th International Conference on Parallel Processing*. Berlin, Heidelberg: Springer-Verlag, 2013. (Euro-Par'13), p. 406–419. ISBN 978-3-642-40046-9. Disponível em: <http://dx.doi.org/10.1007/978-3-642-40047-6_42>.

LIU, J. et al. Panacea: Towards holistic optimization of mapreduce applications. In: *Proceedings of the 10th International Symposium on Code Generation and Optimization*. New York, NY, USA: ACM, 2012. (CGO '12), p. 33–43. ISBN 978-1-4503-1206-6. Disponível em: <http://doi.acm.org/10.1145/2259016.2259022>.

MA, D. yi et al. Registration with linear regression model in augmented reality. In: *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. v. 2, p. 520–523.

MAGOULAS, G. D.; ELDABI, T.; PAUL, R. J. General methodology 3: Global search strategies for simulation optimisation. In: *Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers*. Winter Simulation Conference, 2002. (WSC '02), p. 1978–1985. ISBN 0-7803-7615-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1030453.1030747>.

MELL, P. M.; GRANCE, T. *SP 800-145. The NIST Definition of Cloud Computing.* Gaithersburg, MD, United States, 2011.

MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs (2Nd, Extended Ed.).* New York, NY, USA: Springer-Verlag New York, Inc., 1994. ISBN 3-540-58090-5.

NEURAIS homepage de R. *homepage de Redes Neurais.* 2000. Http://www.din.uem.br/ia/neurais/. Accessed in june 2015.

OPENCV. *OpenCV.* 2011. Http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html. Accessed in June 2015.

OPENSTACK. *OpenStack.* 2015. Https://www.openstack.org/. Accessed in May 2015.

POSK, P.; HUYER, W.; PAL, L. A comparison of global search algorithms for continuous black box optimization. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 20, n. 4, p. 509–541, dez. 2012. ISSN 1063-6560. Disponível em: <http://dx.doi.org/10.1162/EVCO_a_00084>.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach.* 3rd. ed. [S.l.: s.n.], 2009.

SAKR, S. et al. A survey of large scale data management approaches in cloud environments. *Communications Surveys Tutorials, IEEE*, v. 13, n. 3, p. 311–336, 2011. ISSN 1553-877X.

SANTOS, J.; SILVA, F. IGOV: um sistema de integração de dados governamentais. *Revista Brasileira de Administração Científica - Anais do Simpósio Brasileiro de Tecnologia da Informação (SBTI)*, v. 5, n. 2, junho 2014. ISSN 2179-684X.

TANG, S.; LEE, B.-S.; HE, B. Towards economic fairness for big data processing in pay-as-you-go cloud computing. In: *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on.* [S.l.: s.n.], 2014. p. 638–643.

TIWARI, A. et al. A scalable auto-tuning framework for compiler optimization. In: *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing.* Washington, DC, USA: IEEE Computer Society, 2009. (IPDPS '09), p. 1–12. ISBN 978-1-4244-3751-1. Disponível em: <http://dx.doi.org/10.1109/IPDPS.2009.5161054>.

VAQUERO, L. M. et al. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, dez. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1496091.1496100>.

WANG, G. et al. Using realistic simulation for performance analysis of mapreduce setups. In: *Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance.* New York, NY, USA: ACM, 2009. (LSAP '09), p. 19–26. ISBN 978-1-60558-592-5. Disponível em: <http://doi.acm.org/10.1145/1552272.1552278>.

WANG, K.; LIN, X.; TANG, W. Predator - an experience guided configuration optimizer for hadoop mapreduce. In: *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on.* [S.l.: s.n.], 2012. p. 419–426.

WEKA. *Weka*. 2011. Http://www.cs.waikato.ac.nz/ ml/weka/. Accessed in May 2015.

WU, D.; GOKHALE, A. A self-tuning system based on application profiling and performance analysis for optimizing hadoop mapreduce cluster configuration. In: *High Performance Computing (HiPC), 2013 20th International Conference on High Performance Computing*. [S.l.: s.n.], 2013. p. 89–98.

XIAOFANG, Y.; JVLIN, G.; GUODONG, S. Utilization efficiency forecasting of moisture content in maize based on particle swarm optimization algorithm and rbf neural network. In: *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*. [S.l.: s.n.], 2010. v. 4, p. 347–350.

YIGITBASI, N. et al. Towards machine learning-based auto-tuning of mapreduce. In: *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*. [S.l.: s.n.], 2013. p. 11–20. ISSN 1526-7539.

ZHANG, J. pei; LI, Z.-W.; YANG, J. A parallel svm training algorithm on large-scale classification problems. In: *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*. [S.l.: s.n.], 2005. v. 3, p. 1637–1641 Vol. 3.

ZHU, J.; SUN, S. Single-task and multitask sparse gaussian processes. In: *Machine Learning and Cybernetics (ICMLC), 2013 International Conference on*. [S.l.: s.n.], 2013. v. 03, p. 1033–1038.

ZIKOPOULOS, I.; EATON, C.; ZIKOPOULOS, P. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. Mcgraw-hill, 2011. ISBN 9780071790536. Disponível em: <http://books.google.com.br/books?id=Plcg_9NJ_fUC>.

Appendices

A seguir, encontra-se a lista de parâmetros do Hadoop que podem ser ajustados para personalizar a execução das aplicações Hadoop (HADOOP, 2008).

Tabela 6 – Parâmetros do Hadoop que podem ser cofigurados.

Nome	Valor	Descrição
mapreduce. jobhistory. jobtracker. location		If job tracker is static the history files are stored in this single well known place. If No value is set here, by default, it is in the local file system at $hadoop.log.dir/history.
mapreduce. jobtracker. jobhistory. task. numberprogresssplits	12	Every task attempt progresses from 0.0 to 1.0 [unless it fails or is killed]. We record, for each task attempt, certain statistics over each twelfth of the progress range. You can change the number of intervals we divide the entire range of progress into by setting this property. Higher values give more precision to the recorded data, but costs more memory in the job tracker at runtime. Each increment in this attribute costs 16 bytes per running task.
mapreduce. job. userhistorylocation		User can specify a location to store the history files of a particular job. If nothing is specified, the logs are stored in output directory.
mapreduce. jobtracker. jobhistory. completed. location		The completed job history files are stored at this single well known location. If nothing is specified, the files are stored at $mapreduce.jobtracker.jobhistory.location.
mapreduce. job. committer. setup. cleanup. needed	true	true, if job needs job-setup and job-cleanup.false, otherwise
mapreduce. task. io. sort. factor	10	The number of streams to merge at once while sorting files. This determines the number of open file handles.
mapreduce. task. io. sort. mb	100	The total amount of buffer memory to use while sorting files, in megabytes. By default, gives each merge stream 1MB, which should minimize seeks.
mapreduce. map. sort. spill. percent	0.80	The soft limit in the serialization buffer. Once reached, a thread will begin to spill the contents to disk in the background. Note that collection will not block if this threshold is exceeded while a spill is already in progress, so spills may be larger than this threshold when it is set to less than .5
mapreduce. jobtracker. address	local	The host and port that the MapReduce job tracker runs at. If "local", then jobs are run in-process as a single map and reduce task.
mapreduce. local. clientfactory. class. name	org. apache . hadoop . mapred .LocalClientFactory	This the client factory that is responsible for creating local job runner client
mapreduce. jobtracker. http. address	0.0.0.0:50030	The job tracker http server address and port the server will listen on. If the port is 0 then the server will start on a free port.
mapreduce. jobtracker. handler. count	10	The number of server threads for the JobTracker. This should be roughly 4% of the number of tasktracker nodes.
mapreduce. tasktracker. report. address	127.0.0.1:0	The interface and port that task tracker server listens on. Since it is only connected to by the tasks, it uses the local interface. EXPERT ONLY. Should only be changed if your host does not have the loopback interface.
mapreduce. cluster. local. dir	$ hadoop. tmp. dir / mapred/ local	The local directory where MapReduce stores intermediate data files. May be a comma-separated list of directories on different devices in order to spread disk i/o. Directories that do not exist are ignored.
mapreduce. jobtracker. system. dir	$ hadoop. tmp. dir / mapred/ system	The directory where MapReduce stores control files.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. jobtracker. staging. root. dir	$ hadoop. tmp. dir / mapred/ staging	The root of the staging area for users' job files In practice, this should be the directory where users' home directories are located (usually /user)
mapreduce. cluster. temp. dir	$ hadoop. tmp. dir / mapred/ temp	A shared directory for temporary files.
mapreduce. tasktracker. local. dir. minspacestart	0	If the space in mapreduce.cluster.local.dir drops under this, do not ask for more tasks. Value in bytes.
mapreduce. tasktracker. local. dir. minspacekill	0	If the space in mapreduce.cluster.local.dir drops under this, do not ask more tasks until all the current ones have finished and cleaned up. Also, to save the rest of the tasks we have running, kill one of them, to clean up some space. Start with the reduce tasks, then go with the ones that have finished the least. Value in bytes.
mapreduce. jobtracker. expire. trackers. interval	600000	Expert: The time-interval, in miliseconds, after which a tasktracker is declared 'lost' if it doesn't send heartbeats.
mapreduce. tasktracker. instrumentation	org. apache. hadoop. mapred. TaskTrackerMetricsInst	Expert: The instrumentation class to associate with each TaskTracker.
mapreduce. tasktracker. resourcecalculatorplugin		Name of the class whose instance will be used to query resource information on the tasktracker. The class must be an instance of org.apache.hadoop.util.ResourceCalculatorPlugin. If the value is null, the tasktracker attempts to use a class appropriate to the platform. Currently, the only platform supported is Linux.
mapreduce. tasktracker. taskmemorymanager. monitoringinterval	5000	The interval, in milliseconds, for which the tasktracker waits between two cycles of monitoring its tasks' memory usage. Used only if tasks' memory management is enabled via mapred.tasktracker.tasks.maxmemory.
mapreduce. tasktracker. tasks. sleeptimebeforesigkill	5000	The time, in milliseconds, the tasktracker waits for sending a SIGKILL to a task, after it has been sent a SIGTERM. This is currently not used on WINDOWS where tasks are just sent a SIGTERM.
mapreduce. job. maps	2	The default number of map tasks per job. Ignored when mapreduce.jobtracker.address is "local".
mapreduce. job. reduces	1	The default number of reduce tasks per job. Typically set to 99of the cluster's reduce capacity, so that if a node fails the reduces can still be executed in a single wave. Ignored when mapreduce.jobtracker.address is "local".
mapreduce. jobtracker. restart. recover	false	"true" to enable (job) recovery upon restart, "false" to start afresh.
mapreduce. jobtracker. jobhistory. block. size	3145728	The block size of the job history file. Since the job recovery uses job history, its important to dump job history to disk as soon as possible. Note that this is an expert level parameter. The default value is set to 3 MB.
mapreduce. jobtracker. taskscheduler	org. apache. hadoop. mapred. JobQueueTaskScheduler	The class responsible for scheduling the tasks.
mapreduce. job. reducer. preempt. delay. sec	0	The threshold in terms of seconds after which an unsatisfied mapper request triggers reducer preemption to free space. Default 0 implies that the reduces should be preempted immediately after allocation if there is currently no room for newly allocated mappers.
mapreduce. job. max. split. locations	10	The max number of block locations to store for each split for locality calculation.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. job. split. metainfo. maxsize	10000000	The maximum permissible size of the split metainfo file. The JobTracker won't attempt to read split metainfo files bigger than the configured value. No limits if set to -1.
mapreduce. jobtracker. taskscheduler. maxrunningtasks. perjob		The maximum number of running tasks for a job before it gets preempted. No limits if undefined.
mapreduce. map. maxattempts	4	Expert: The maximum number of attempts per map task. In other words, framework will try to execute a map task these many number of times before giving up on it.
mapreduce. reduce. maxattempts	4	Expert: The maximum number of attempts per reduce task. In other words, framework will try to execute a reduce task these many number of times before giving up on it.
mapreduce. reduce. shuffle. fetch. retry. enabled	$ yarn. nodemanager. recovery. enabled	Set to enable fetch retry during host restart.
mapreduce. reduce. shuffle. fetch. retry. interval-ms	1000	Time of interval that fetcher retry to fetch again when some non-fatal failure happens because of some events like NM restart.
mapreduce. reduce. shuffle. fetch. retry. timeout-ms	30000	Timeout value for fetcher to retry to fetch again when some non-fatal failure happens because of some events like NM restart.
mapreduce. reduce. shuffle. retry-delay. max. ms	60000	The maximum number of ms the reducer will delay before retrying to download map data.
mapreduce. reduce. shuffle. parallelcopies	5	The default number of parallel transfers run by reduce during the copy(shuffle) phase.
mapreduce. reduce. shuffle. connect. timeout	180000	Expert: The maximum amount of time (in milli seconds) reduce task spends in trying to connect to a tasktracker for getting map output.
mapreduce. reduce. shuffle. read. timeout	180000	Expert: The maximum amount of time (in milli seconds) reduce task waits for map output data to be available for reading after obtaining connection.
mapreduce. shuffle. connection-keep-alive. enable	false	set to true to support keep-alive connections.
mapreduce. shuffle. connection-keep-alive. timeout	5	The number of seconds a shuffle client attempts to retain http connection. Refer "Keep-Alive: timeout="header in Http specification
mapreduce. task. timeout	600000	The number of milliseconds before a task will be terminated if it neither reads an input, writes an output, nor updates its status string. A value of 0 disables the timeout.
mapreduce. tasktracker. map. tasks. maximum	2	The maximum number of map tasks that will be run simultaneously by a task tracker.
mapreduce. tasktracker. reduce. tasks. maximum	2	The maximum number of reduce tasks that will be run simultaneously by a task tracker.
mapreduce. map. memory. mb	1024	The amount of memory to request from the scheduler for each map task.
mapreduce. map. cpu. vcores	1	The number of virtual cores to request from the scheduler for each map task.
mapreduce. reduce. memory. mb	1024	The amount of memory to request from the scheduler for each reduce task.
mapreduce. reduce. cpu. vcores	1	The number of virtual cores to request from the scheduler for each reduce task.
mapreduce. jobtracker. retiredjobs. cache.size	1000	The number of retired job status to keep in the cache.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. tasktracker. outofband. heartbeat	false	Expert: Set this to true to let the tasktracker send an out-of-band heartbeat on task-completion for better latency.
mapreduce. jobtracker. jobhistory. lru. cache. size	5	The number of job history files loaded in memory. The jobs are loaded when they are first accessed. The cache is cleared based on LRU.
mapreduce. jobtracker. instrumentation	org. apache. hadoop. mapred. JobTrackerMetricsInst	Expert: The instrumentation class to associate with each JobTracker.
mapred. child. java. opts	-Xmx200m	Java opts for the task processes. The following symbol, if present, will be interpolated: @taskid@ is replaced by current TaskID. Any other occurrences of '@' will go unchanged. For example, to enable verbose gc logging to a file named for the taskid in /tmp and to set the heap maximum to be a gigabyte, pass a 'value' of: -Xmx1024m -verbose:gc -Xloggc:/tmp/@taskid@.gc Usage of -Djava.library.path can cause programs to no longer function if hadoop native libraries are used. These values should instead be set as part of LD_LIBRARY_PATH in the map / reduce JVM env using the mapreduce.map.env and mapreduce.reduce.env config settings.
mapred. child. env		User added environment variables for the task processes. Example : 1) A=foo This will set the env variable A to foo 2) B=$B:c This is inherit nodemanager's B env variable on Unix. 3) B=%B%;c This is inherit nodemanager's B env variable on Windows.
mapreduce. admin. user. env		Expert: Additional execution environment entries for map and reduce task processes. This is not an additive property. You must preserve the original value if you want your map and reduce tasks to have access to native libraries (compression, etc). When this value is empty, the command to set execution envrionment will be OS dependent.
mapreduce. task. tmp. dir	./tmp	To set the value of tmp directory for map and reduce tasks. If the value is an absolute path, it is directly assigned. Otherwise, it is prepended with task's working directory. The java tasks are executed with option -Djava.io.tmpdir='the absolute path of the tmp dir'. Pipes and streaming are set with environment variable, TMPDIR='the absolute path of the tmp dir'
mapreduce. map. log. level	INFO	The logging level for the map task. The allowed levels are: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE and ALL.
mapreduce. reduce. log. level	INFO	The logging level for the reduce task. The allowed levels are: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE and ALL.
mapreduce. map. cpu. vcores	1	The number of virtual cores required for each map task.
mapreduce. reduce. cpu. vcores	1	The number of virtual cores required for each reduce task.
mapreduce. reduce. merge. inmem. threshold	1000	The threshold, in terms of the number of files for the in-memory merge process. When we accumulate threshold number of files we initiate the in-memory merge and spill to disk. A value of 0 or less than 0 indicates we want to DON'T have any threshold and instead depend only on the ramfs's memory consumption to trigger the merge.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. reduce. shuffle. merge. percent	0.66	The usage threshold at which an in-memory merge will be initiated, expressed as a percentage of the total memory allocated to storing in-memory map outputs, as defined by mapreduce.reduce.shuffle.input.buffer.percent.
mapreduce. reduce. shuffle. input. buffer. percent	0.70	The percentage of memory to be allocated from the maximum heap size to storing map outputs during the shuffle.
mapreduce. reduce. input. buffer. percent	0.0	The percentage of memory- relative to the maximum heap size- to retain map outputs during the reduce. When the shuffle is concluded, any remaining map outputs in memory must consume less than this threshold before the reduce can begin.
mapreduce. reduce. shuffle. memory. limit. percent	0.25	Expert: Maximum percentage of the in-memory limit that a single shuffle can consume
mapreduce. shuffle. ssl. enabled	false	Whether to use SSL for for the Shuffle HTTP endpoints.
mapreduce. shuffle. ssl. file. buffer. size	65536	Buffer size for reading spills from file when using SSL.
mapreduce. shuffle. max. connections	0	Max allowed connections for the shuffle. Set to 0 (zero) to indicate no limit on the number of connections.
mapreduce. shuffle. max. threads	0	Max allowed threads for serving shuffle connections. Set to zero to indicate the default of 2 times the number of available processors (as reported by Runtime.availableProcessors()). Netty is used to serve requests, so a thread is not needed for each connection.
mapreduce. shuffle. transferTo. allowed		This option can enable/disable using nio transferTo method in the shuffle phase. NIO transferTo does not perform well on windows in the shuffle phase. Thus, with this configuration property it is possible to disable it, in which case custom transfer method will be used. Recommended value is false when running Hadoop on Windows. For Linux, it is recommended to set it to true. If nothing is set then the default value is false for Windows, and true for Linux.
mapreduce. shuffle. transfer. buffer. size	131072	This property is used only if mapreduce.shuffle.transferTo.allowed is set to false. In that case, this property defines the size of the buffer used in the buffer copy code for the shuffle phase. The size of this buffer determines the size of the IO requests.
mapreduce. reduce. markreset. buffer. percent	0.0	The percentage of memory -relative to the maximum heap size- to be used for caching values when using the mark-reset functionality.
mapreduce. map. speculative	true	If true, then multiple instances of some map tasks may be executed in parallel.
mapreduce. reduce. speculative	true	If true, then multiple instances of some reduce tasks may be executed in parallel.
mapreduce. job. speculative. speculativecap	0.1	The max percent (0-1) of running tasks that can be speculatively re-executed at any time.
mapreduce. job. map. output. collector. class	org. apache. hadoop. mapred. MapTask$MapOutputBuffer	The MapOutputCollector implementation(s) to use. This may be a comma-separated list of class names, in which case the map task will try to initialize each of the collectors in turn. The first to successfully initialize will be used.
mapreduce. job. speculative. slowtaskthreshold	1.0	The number of standard deviations by which a task's ave progress-rates must be lower than the average of all running tasks' for the task to be considered too slow.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. job. speculative. slownodethreshold	1.0	The number of standard deviations by which a Task Tracker's ave map and reduce progress-rates (finishTime-dispatchTime) must be lower than the average of all successful map/reduce task's for the TT to be considered too slow to give a speculative task to.
mapreduce. job. jvm. numtasks	1	How many tasks to run per jvm. If set to -1, there is no limit.
mapreduce. job. ubertask. enable	false	Whether to enable the small-jobs "ubertask"optimization, which runs "sufficiently small"jobs sequentially within a single JVM. "Small"is defined by the following maxmaps, maxreduces, and maxbytes settings. Note that configurations for application masters also affect the "Small" definition - yarn.app.mapreduce.am.resource.mb must be larger than both mapreduce.map.memory.mb and mapreduce.reduce.memory.mb, and yarn.app.mapreduce.am.resource.cpu-vcores must be larger than both mapreduce.map.cpu.vcores and mapreduce.reduce.cpu.vcores to enable ubertask. Users may override this value.
mapreduce. job. ubertask. maxmaps	9	Threshold for number of maps, beyond which job is considered too big for the ubertasking optimization. Users may override this value, but only downward.
mapreduce. job. ubertask. maxreduces	1	Threshold for number of reduces, beyond which job is considered too big for the ubertasking optimization. CURRENTLY THE CODE CANNOT SUPPORT MORE THAN ONE REDUCE and will ignore larger values. (Zero is a valid max, however.) Users may override this value, but only downward.
mapreduce. job. ubertask. maxbytes		Threshold for number of input bytes, beyond which job is considered too big for the ubertasking optimization. If no value is specified, dfs.block.size is used as a default. Be sure to specify a default value in mapred-site.xml if the underlying filesystem is not HDFS. Users may override this value, but only downward.
mapreduce. job. emittimeline-data	false	Specifies if the Application Master should emit timeline data to the timeline server. Individual jobs can override this value.
mapreduce. input. fileinputformat. split. minsize	0	The minimum size chunk that map input should be split into. Note that some file formats may have minimum split sizes that take priority over this setting.
mapreduce. input. fileinputformat. liststatus. num-threads	1	The number of threads to use to list and fetch block locations for the specified input paths. Note: multiple threads should not be used if a custom non thread-safe path filter is used.
mapreduce. jobtracker. maxtasks. perjob	-1	The maximum number of tasks for a single job. A value of -1 indicates that there is no maximum.
mapreduce. input. lineinputformat. linespermap	1	When using NLineInputFormat, the number of lines of input data to include in each split.
mapreduce. client. submit. file. replication	10	The replication level for submitted job files. This should be around the square root of the number of nodes.
mapreduce. tasktracker. dns. interface	default	The name of the Network Interface from which a task tracker should report its IP address.
mapreduce. tasktracker. dns. nameserver	default	The host name or IP address of the name server (DNS) which a TaskTracker should use to determine the host name used by the JobTracker for communication and display purposes.
mapreduce. tasktracker. http. threads	40	The number of worker threads that for the http server. This is used for map output fetching

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. tasktracker. http. address	0.0.0.0:50060	The task tracker http server address and port. If the port is 0 then the server will start on a free port.
mapreduce. task. files. preserve. failedtasks	false	Should the files for failed tasks be kept. This should only be used on jobs that are failing, because the storage is never reclaimed. It also prevents the map outputs from being erased from the reduce directory as they are consumed.
mapreduce. output. fileoutputformat. compress	false	Should the job outputs be compressed?
mapreduce. output. fileoutputformat. compress.type	RECORD	If the job outputs are to compressed as SequenceFiles, how should they be compressed? Should be one of NONE, RECORD or BLOCK.
mapreduce. output. fileoutputformat. compress. codec	org. apache. hadoop. io. compress. DefaultCodec	If the job outputs are compressed, how should they be compressed?
mapreduce.map.output.compress	false	Should the outputs of the maps be compressed before being sent across the network. Uses SequenceFile compression.
mapreduce. map. output. compress. codec	org. apache. hadoop. io. compress. DefaultCodec	If the map outputs are compressed, how should they be compressed?
map. sort. class	org. apache. hadoop. util. QuickSort	The default sort class for sorting keys.
mapreduce. task. userlog. limit. kb	0	The maximum size of user-logs of each task in KB. 0 disables the cap.
yarn. app. mapreduce. am. container. log. limit.kb	0	The maximum size of the MRAppMaster attempt container logs in KB. 0 disables the cap.
yarn. app. mapreduce. task. container. log. backups	0	Number of backup files for task logs when using ContainerRollingLogAppender (CRLA). See org.apache.log4j.RollingFileAppender.maxBackupIndex. By default, ContainerLogAppender (CLA) is used, and container logs are not rolled. CRLA is enabled for tasks when both mapreduce.task.userlog.limit.kb and yarn.app.mapreduce.task.container.log.backups are greater than zero.
yarn. app. mapreduce. am. container. log. backups	0	Number of backup files for the ApplicationMaster logs when using ContainerRollingLogAppender (CRLA). See org.apache.log4j.RollingFileAppender.maxBackupIndex. By default, ContainerLogAppender (CLA) is used, and container logs are not rolled. CRLA is enabled for the ApplicationMaster when both mapreduce.task.userlog.limit.kb and yarn.app.mapreduce.am.container.log.backups are greater than zero.
mapreduce. job. userlog. retain. hours	24	The maximum time, in hours, for which the user-logs are to be retained after the job completion.
mapreduce. jobtracker. hosts. filename		Names a file that contains the list of nodes that may connect to the jobtracker. If the value is empty, all hosts are permitted.
mapreduce. jobtracker. hosts. exclude. filename		Names a file that contains the list of hosts that should be excluded by the jobtracker. If the value is empty, no hosts are excluded.
mapreduce. jobtracker. heartbeats. in. second	100	Expert: Approximate number of heart-beats that could arrive at JobTracker in a second. Assuming each RPC can be processed in 10msec, the default value is made 100 RPCs in a second.
mapreduce. jobtracker. tasktracker. maxblacklists	4	The number of blacklists for a taskTracker by various jobs after which the task tracker could be blacklisted across all jobs. The tracker will be given a tasks later (after a day). The tracker will become a healthy tracker after a restart.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. job. maxtaskfailures. per. tracker	3	The number of task-failures on a tasktracker of a given job after which new tasks of that job aren't assigned to it. It MUST be less than mapreduce.map.maxattempts and mapreduce.reduce.maxattempts otherwise the failed task will never be tried on a different node.
mapreduce. client. output. filter	FAILED	The filter for controlling the output of the task's userlogs sent to the console of the JobClient. The permissible options are: NONE, KILLED, FAILED, SUCCEEDED and ALL.
mapreduce. client. completion. pollinterval	5000	The interval (in milliseconds) between which the JobClient polls the JobTracker for updates about job status. You may want to set this to a lower value to make tests run faster on a single node system. Adjusting this value in production may lead to unwanted client-server traffic.
mapreduce. client. progressmonitor. pollinterval	1000	The interval (in milliseconds) between which the JobClient reports status to the console and checks for job completion. You may want to set this to a lower value to make tests run faster on a single node system. Adjusting this value in production may lead to unwanted client-server traffic.
mapreduce. jobtracker. persist. jobstatus. active	true	Indicates if persistency of job status information is active or not.
mapreduce. jobtracker. persist. jobstatus. hours	1	The number of hours job status information is persisted in DFS. The job status information will be available after it drops of the memory queue and between jobtracker restarts. With a zero value the job status information is not persisted at all in DFS.
mapreduce. jobtracker. persist. jobstatus. dir	/jobtracker/jobsInfo	The directory where the job status information is persisted in a file system to be available after it drops of the memory queue and between jobtracker restarts.
mapreduce. task. profile	false	To set whether the system should collect profiler information for some of the tasks in this job? The information is stored in the user log directory. The value is "true"if task profiling is enabled.
mapreduce. task. profile. maps	0-2	To set the ranges of map tasks to profile. mapreduce.task.profile has to be set to true for the value to be accounted.
mapreduce. task. profile. reduces	0-2	To set the ranges of reduce tasks to profile. mapreduce.task.profile has to be set to true for the value to be accounted.
mapreduce. task. profile. params	-agentlib:hprof=cpu =samples, heap=sites, force=n, thread=y,verbose=n ,file=%s	JVM profiler parameters used to profile map and reduce task attempts. This string may contain a single format specifier %s that will be replaced by the path to profile.out in the task attempt log directory. To specify different profiling options for map tasks and reduce tasks, more specific parameters mapreduce.task.profile.map.params and mapreduce.task.profile.reduce.params should be used.
mapreduce. task. profile. map. params	$ mapreduce. task. profile. params	Map-task-specific JVM profiler parameters. See mapreduce. task. profile. params
mapreduce. task. profile. reduce. params	$ mapreduce. task. profile. params	Reduce-task-specific JVM profiler parameters. See mapreduce.task.profile.params
mapreduce. task. skip. start. attempts	2	The number of Task attempts AFTER which skip mode will be kicked off. When skip mode is kicked off, the tasks reports the range of records which it will process next, to the TaskTracker. So that on failures, TT knows which ones are possibly the bad records. On further executions, those are skipped.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. map. skip. proc. count. autoincr	true	The flag which if set to true, SkipBadRecords.COUNTER_MAP_PROCESSED_RECORDS is incremented by MapRunner after invoking the map function. This value must be set to false for applications which process the records asynchronously or buffer the input records. For example streaming. In such cases applications should increment this counter on their own.
mapreduce. reduce. skip. proc. count. autoincr	true	The flag which if set to true, SkipBadRecords.COUNTER_REDUCE_PROCESSED_GROUPS is incremented by framework after invoking the reduce function. This value must be set to false for applications which process the records asynchronously or buffer the input records. For example streaming. In such cases applications should increment this counter on their own.
mapreduce. job. skip. outdir		If no value is specified here, the skipped records are written to the output directory at _logs/skip. User can stop writing skipped records by giving the value "none".
mapreduce. map. skip. maxrecords	0	The number of acceptable skip records surrounding the bad record PER bad record in mapper. The number includes the bad record as well. To turn the feature of detection/skipping of bad records off, set the value to 0. The framework tries to narrow down the skipped range by retrying until this threshold is met OR all attempts get exhausted for this task. Set the value to Long.MAX_VALUE to indicate that framework need not try to narrow down. Whatever records(depends on application) get skipped are acceptable.
mapreduce. reduce. skip. maxgroups	0	The number of acceptable skip groups surrounding the bad group PER bad group in reducer. The number includes the bad group as well. To turn the feature of detection/skipping of bad groups off, set the value to 0. The framework tries to narrow down the skipped range by retrying until this threshold is met OR all attempts get exhausted for this task. Set the value to Long.MAX_VALUE to indicate that framework need not try to narrow down. Whatever groups(depends on application) get skipped are acceptable.
mapreduce. ifile. readahead	true	Configuration key to enable/disable IFile readahead.
mapreduce. ifile. readahead. bytes	4194304	Configuration key to set the IFile readahead length in bytes.
mapreduce. jobtracker. taskcache. levels	2	This is the max level of the task cache. For example, if the level is 2, the tasks cached are at the host level and at the rack level.
mapreduce. job. queuename	default	Queue to which a job is submitted. This must match one of the queues defined in mapred-queues.xml for the system. Also, the ACL setup for the queue must allow the current user to submit a job to the queue. Before specifying a queue, ensure that the system is configured with the queue, and access is allowed for submitting jobs to the queue.
mapreduce. job. tags		Tags for the job that will be passed to YARN at submission time. Queries to YARN for applications can filter on these tags.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. cluster. acls. enabled	false	Specifies whether ACLs should be checked for authorization of users for doing various queue and job level operations. ACLs are disabled by default. If enabled, access control checks are made by JobTracker and TaskTracker when requests are made by users for queue operations like submit job to a queue and kill a job in the queue and job operations like viewing the job-details (See mapreduce.job.acl-view-job) or for modifying the job (See mapreduce.job.acl-modify-job) using Map/Reduce APIs, RPCs or via the console and web user interfaces. For enabling this flag(mapreduce.cluster.acls.enabled), this is to be set to true in mapred-site.xml on JobTracker node and on all TaskTracker nodes.
mapreduce. job. acl-modify-job		Job specific access-control list for 'modifying' the job. It is only used if authorization is enabled in Map/Reduce by setting the configuration property mapreduce.cluster.acls.enabled to true. This specifies the list of users and/or groups who can do modification operations on the job. For specifying a list of users and groups the format to use is "user1,user2 group1,group". If set to '*', it allows all users/groups to modify this job. If set to ' '(i.e. space), it allows none. This configuration is used to guard all the modifications with respect to this job and takes care of all the following operations: o killing this job o killing a task of this job, failing a task of this job o setting the priority of this job Each of these operations are also protected by the per-queue level ACL "acl-administer-jobs" configured via mapred-queues.xml. So a caller should have the authorization to satisfy either the queue-level ACL or the job-level ACL. Irrespective of this ACL configuration, (a) job-owner, (b) the user who started the cluster, (c) members of an admin configured supergroup configured via mapreduce.cluster.permissions.supergroup and (d) queue administrators of the queue to which this job was submitted to configured via acl-administer-jobs for the specific queue in mapred-queues.xml can do all the modification operations on a job. By default, nobody else besides job-owner, the user who started the cluster, members of supergroup and queue administrators can perform modification operations on a job.
		Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. job. acl-view-job		Job specific access-control list for 'viewing' the job. It is only used if authorization is enabled in Map/Reduce by setting the configuration property mapreduce.cluster.acls.enabled to true. This specifies the list of users and/or groups who can view private details about the job. For specifying a list of users and groups the format to use is "user1,user2 group1,group". If set to '*', it allows all users/groups to modify this job. If set to ' '(i.e. space), it allows none. This configuration is used to guard some of the job-views and at present only protects APIs that can return possibly sensitive information of the job-owner like o job-level counters o task-level counters o tasks' diagnostic information o task-logs displayed on the TaskTracker web-UI and o job.xml showed by the JobTracker's web-UI Every other piece of information of jobs is still accessible by any other user, for e.g., JobStatus, JobProfile, list of jobs in the queue, etc. Irrespective of this ACL configuration, (a) job-owner, (b) the user who started the cluster, (c) members of an admin configured supergroup configured via mapreduce.cluster.permissions.supergroup and (d) queue administrators of the queue to which this job was submitted to configured via acl-administer-jobs for the specific queue in mapred-queues.xml can do all the view operations on a job. By default, nobody else besides job-owner, the user who started the cluster, memebers of supergroup and queue administrators can perform view operations on a job.
mapreduce. tasktracker. indexcache. mb	10	The maximum memory that a task tracker allows for the index cache that is used when serving map outputs to reducers.
mapreduce. job. token. tracking. ids. enabled	false	Whether to write tracking ids of tokens to job-conf. When true, the configuration property "mapreduce. job. token. tracking. ids" is set to the token-tracking-ids of the job
mapreduce. job. token. tracking. ids		When mapreduce.job.token.tracking.ids.enabled is set to true, this is set by the framework to the token-tracking-ids used by the job.
mapreduce. task. merge. progress. records	10000	The number of records to process during merge before sending a progress notification to the TaskTracker.
mapreduce. task. combine. progress. records	10000	The number of records to process during combine output collection before sending a progress notification.
mapreduce. job. reduce. slowstart. completedmaps	0.05	Fraction of the number of maps in the job which should be complete before reduces are scheduled for the job.
mapreduce. job. complete. cancel. delegation. tokens	true	if false - do not unregister/cancel delegation tokens from renewal, because same tokens may be used by spawned jobs
mapreduce. tasktracker. taskcontroller	org. apache. hadoop. mapred. DefaultTaskController	TaskController which is used to launch and manage task execution
mapreduce. tasktracker. group		Expert: Group to which TaskTracker belongs. If LinuxTaskController is configured via mapreduce.tasktracker.taskcontroller, the group owner of the task-controller binary should be same as this group.
mapreduce.shuffle.port	13562	Default port that the ShuffleHandler will run on. ShuffleHandler is a service run at the NodeManager to facilitate transfers of intermediate Map outputs to requesting Reducers.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. job. reduce. shuffle. consumer. plugin. class	org. apache. hadoop. mapreduce. task. reduce. Shuffle	Name of the class whose instance will be used to send shuffle requests by reducetasks of this job. The class must be an instance of org. apache. hadoop. mapred. ShuffleConsumerPlugin.
mapreduce. tasktracker. healthchecker. script. path		Absolute path to the script which is periodicallyrun by the node health monitoring service to determine if the node is healthy or not. If the value of this key is empty or the file does not exist in the location configured here, the node health monitoring service is not started.
mapreduce. tasktracker. healthchecker. interval	60000	Frequency of the node health script to be run, in milliseconds
mapreduce. tasktracker. healthchecker. script. timeout	600000	Time after node health script should be killed if unresponsive and considered that the script has failed.
mapreduce. tasktracker. healthchecker. script. args		List of arguments which are to be passed to node health script when it is being launched comma seperated.
mapreduce. job. counters. limit	120	Limit on the number of user counters allowed per job.
hline mapreduce. framework. name	local	The runtime framework for executing MapReduce jobs. Can be one of local, classic or yarn.
yarn. app. mapreduce. am. staging-dir	/tmp/hadoop-yarn/staging	The staging dir used while submitting jobs.
mapreduce. am. maxattempts	2	The maximum number of application attempts. It is a application-specific setting. It should not be larger than the global number set by resourcemanager. Otherwise, it will be override. The default number is set to 2, to allow at least one retry for AM.
mapreduce. job. end-notification. url	http://localhost:8080/jobstatus.php?jobId=$jobId&jobStatus=$jobStatus	Indicates url which will be called on completion of job to inform end status of job. User can give at most 2 variables with URI : $jobId and $jobStatus. If they are present in URI, then they will be replaced by their respective values.
mapreduce. job. end-notification. retry. attempts	0	The number of times the submitter of the job wants to retry job end notification if it fails. This is capped by mapreduce. job. end-notification. max. attempts
mapreduce. job. end-notification. retry. interval	1000	The number of milliseconds the submitter of the job wants to wait before job end notification is retried if it fails. This is capped by mapreduce. job. end-notification. max. retry. interval
mapreduce. job. end-notification. max.attempts	5	The maximum number of times a URL will be read for providing job end notification. Cluster administrators can set this to limit how long after end of a job, the Application Master waits before exiting. Must be marked as final to prevent users from overriding this.
mapreduce. job. end-notification. max. retry. interval	5000	The maximum amount of time (in milliseconds) to wait before retrying job end notification. Cluster administrators can set this to limit how long the Application Master waits before exiting. Must be marked as final to prevent users from overriding this.
yarn. app. mapreduce. am. env		User added environment variables for the MR App Master processes. Example : 1) A=foo This will set the env variable A to foo 2) B=$B:c This is inherit tasktracker's B env variable.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
yarn. app. mapreduce. am. admin. user. env		Environment variables for the MR App Master processes for admin purposes. These values are set first and can be overridden by the user env (yarn.app.mapreduce.am.env) Example : 1) A=foo This will set the env variable A to foo 2) B=$B:c This is inherit app master's B env variable.
yarn. app. mapreduce. am. command-opts	-Xmx1024m	Java opts for the MR App Master processes. The following symbol, if present, will be interpolated: @taskid@ is replaced by current TaskID. Any other occurrences of '@' will go unchanged. For example, to enable verbose gc logging to a file named for the taskid in /tmp and to set the heap maximum to be a gigabyte, pass a 'value' of: -Xmx1024m -verbose:gc -Xloggc:/tmp/@taskid@.gc Usage of -Djava.library.path can cause programs to no longer function if hadoop native libraries are used. These values should instead be set as part of LD_LIBRARY_PATH in the map / reduce JVM env using the mapreduce. map. env and mapreduce. reduce. env config settings.
yarn. app. mapreduce. am. admin-command-opts		Java opts for the MR App Master processes for admin purposes. It will appears before the opts set by yarn.app.mapreduce.am.command-opts and thus its options can be overridden user. Usage of -Djava.library.path can cause programs to no longer function if hadoop native libraries are used. These values should instead be set as part of LD_LIBRARY_PATH in the map / reduce JVM env using the mapreduce. map. env and mapreduce. reduce. env config settings.
yarn. app. mapreduce. am. job. task. listener. thread-count	30	The number of threads used to handle RPC calls in the MR AppMaster from remote tasks
yarn. app. mapreduce. am. job. client. port-range		Range of ports that the MapReduce AM can use when binding. Leave blank if you want all possible ports. For example 50000-50050,50100-50200
yarn. app. mapreduce. am. job. committer. cancel-timeout	60000	The amount of time in milliseconds to wait for the output committer to cancel an operation if the job is killed
yarn. app. mapreduce. am. job. committer. commit-window	10000	Defines a time window in milliseconds for output commit operations. If contact with the RM has occurred within this window then commits are allowed, otherwise the AM will not allow output commits until contact with the RM has been re-established.
yarn. app. mapreduce. am. scheduler. heartbeat. interval-ms	1000	The interval in ms at which the MR AppMaster should send heartbeats to the ResourceManager
yarn. app. mapreduce. client-am. ipc. max-retries	3	The number of client retries to the AM - before reconnecting to the RM to fetch Application Status.
yarn. app. mapreduce. client-am. ipc. max-retries-on-timeouts	3	The number of client retries on socket timeouts to the AM - before reconnecting to the RM to fetch Application Status.
yarn. app. mapreduce. client. max-retries	3	The number of client retries to the RM/HS before throwing exception. This is a layer above the ipc.
yarn. app. mapreduce. am. resource. mb	1536	The amount of memory the MR AppMaster needs.
yarn. app. mapreduce. am. resource. cpu-vcores	1	The number of virtual CPU cores the MR AppMaster needs.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. application. classpath		CLASSPATH for MR applications. A comma-separated list of CLASSPATH entries. If mapreduce.application.framework is set then this must specify the appropriate classpath for that archive, and the name of the archive must be present in the classpath. If mapreduce.app-submission.cross-platform is false, platform-specific environment vairable expansion syntax would be used to construct the default CLASSPATH entries. Parameter expansion marker will be replaced by NodeManager on container launch based on the underlying OS accordingly.
mapreduce. app-submission. cross-platform	false	If enabled, user can submit an application cross-platform i.e. submit an application from a Windows client to a Linux/Unix server or vice versa.
mapreduce. application. framework. path		Path to the MapReduce framework archive. If set, the framework archive will automatically be distributed along with the job, and this path would normally reside in a public location in an HDFS filesystem. As with distributed cache files, this can be a URL with a fragment specifying the alias to use for the archive name. For example, hdfs:/mapred/framework/hadoop-mapreduce-2.1.1.tar.gz#mrframework would alias the localized archive as "mrframework". Note that mapreduce.application.classpath must include the appropriate classpath for the specified framework. The base name of the archive, or alias of the archive if an alias is used, must appear in the specified classpath.
mapreduce. job. classloader	false	Whether to use a separate (isolated) classloader for user classes in the task JVM.
mapreduce. job. classloader. system. classes		Used to override the default definition of the system classes for the job classloader. The system classes are a comma-separated list of classes that should be loaded from the system classpath, not the user-supplied JARs, when mapreduce.job.classloader is enabled. Names ending in '.' (period) are treated as package names, and names starting with a '-' are treated as negative matches.
mapreduce. jobhistory. address	0.0.0.0:10020	MapReduce JobHistory Server IPC host:port</description>
mapreduce. jobhistory. webapp. address	0.0.0.0:19888	MapReduce JobHistory Server Web UI host:port
mapreduce. jobhistory. keytab	/etc/ security/ keytab /jhs. service. keytab	Location of the kerberos keytab file for the MapReduce JobHistory Server.
mapreduce. jobhistory. principal	jhs/_HOST@REALM. TLD	Kerberos principal name for the MapReduce JobHistory Server.
mapreduce. jobhistory. intermediate-done-dir	$ yarn. app. mapreduce. am. staging-dir / history/ done_intermediate	
mapreduce. jobhistory. done-dir	$ yarn. app. mapreduce. am. staging-dir/history/ done	
mapreduce. jobhistory. cleaner. enable	true	
mapreduce. jobhistory. cleaner. interval-ms	86400000	How often the job history cleaner checks for files to delete, in milliseconds. Defaults to 86400000 (one day). Files are only deleted if they are older than mapreduce.jobhistory.max-age-ms.

Continua na próxima página

Tabela 6 – Continua na próxima página

Nome	Valor	Descrição
mapreduce. jobhistory. max-age-ms	604800000	Job history files older than this many milliseconds will be deleted when the history cleaner runs. Defaults to 604800000 (1 week).
mapreduce. jobhistory. client. thread-count	10	The number of threads to handle client API requests
mapreduce. jobhistory. datestring. cache. size	200000	Size of the date string cache. Effects the number of directories which will be scanned to find a job.
mapreduce. jobhistory. joblist. cache. size	20000	Size of the job list cache
mapreduce. jobhistory. loadedjobs. cache. size	5	Size of the loaded job cache
mapreduce. jobhistory. move. interval-ms	180000	Scan for history files to more from intermediate done dir to done dir at this frequency.
mapreduce. jobhistory. move. thread-count	3	The number of threads used to move files.
mapreduce. jobhistory. store. class		The HistoryStorage class to use to cache history data.
mapreduce. jobhistory. minicluster. fixed. ports	false	Whether to use fixed ports with the minicluster
mapreduce. jobhistory. admin. address	0.0.0.0:10033	The address of the History server admin interface.
mapreduce. jobhistory. admin. acl	*	ACL of who can be admin of the History server.
mapreduce. jobhistory. recovery. enable	false	Enable the history server to store server state and recover server state upon startup. If enabled then mapreduce. jobhistory. recovery. store. class must be specified.
mapreduce. jobhistory. recovery. store. class	org. apache. hadoop. mapreduce. v2. hs. HistoryServerFileSystemStateStoreService	The HistoryServerStateStoreService class to store history server state for recovery.
mapreduce. jobhistory. recovery. store. fs. uri	$ hadoop. tmp. dir/-mapred/ history/ recoverystore	The URI where history server state will be stored if HistoryServerFileSystemStateStoreService is configured as the recovery storage class.
mapreduce. jobhistory. http. policy	HTTP_ONLY	This configures the HTTP endpoint for JobHistoryServer web UI. The following values are supported: - HTTP_ONLY : Service is provided only on http - HTTPS_ONLY : Service is provided only on https

Ao executar o Hadoop repetidas vezes com configurações distintas, foi criado um conjunto de treinamento no intuito de ser a entrada para a criação dos modelos de Função de base radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP) e Perceptron de Múltiplas Camadas (MLP) no Weka para prever o tempo de execução para as configurações pertencente ao intervalo pertencente ao espaço de busca especificado no Wordcount.

Tabela 7 – Conjunto de treinamento obtido para Wordcount.

Número de *mappers*	Número de *reducers*	Taxa de *slowstart*	Tempo de execução (s)
1	1	0.05	243
1	2	0.05	235

Continua na próxima página

Tabela 7 – Continua na próxima página

Número de *mappers*	Número de *reducers*	Taxa de *slowstart*	Tempo de execução (s)
1	4	0.05	245
1	6	0.05	247
1	8	0.05	246
2	1	0.05	240
2	2	0.05	243
2	4	0.05	254
2	6	0.05	253
2	8	0.05	253
4	1	0.05	259
4	2	0.05	264
4	4	0.05	280
4	6	0.05	279
4	8	0.05	267
6	1	0.05	276
6	2	0.05	277
6	4	0.05	287
6	6	0.05	288
6	8	0.05	292
8	1	0.05	290
8	2	0.05	294
8	4	0.05	298
8	6	0.05	302
8	8	0.05	224
1	1	0.2	235
1	2	0.2	238
1	4	0.2	247
1	6	0.2	245
1	8	0.2	246
2	1	0.2	240
2	2	0.2	239
2	4	0.2	248
2	6	0.2	249
2	8	0.2	248
4	1	0.2	260
4	2	0.2	266
4	4	0.2	278
4	6	0.2	282
4	8	0.2	288
6	1	0.2	279
6	2	0.2	291
6	4	0.2	299
6	6	0.2	298
6	8	0.2	288
8	1	0.2	229
8	2	0.2	227
8	4	0.2	317
8	6	0.2	319
8	8	0.2	314
1	1	0.4	242
1	2	0.4	243
1	4	0.4	247
1	6	0.4	244
1	8	0.4	253
2	1	0.4	250
2	2	0.4	250

Continua na próxima página

Tabela 7 – Continua na próxima página

Número de *mappers*	Número de *reducers*	Taxa de *slowstart*	Tempo de execução (s)
2	4	0.4	253
2	6	0.4	264
2	8	0.4	251
4	1	0.4	269
4	2	0.4	277
4	4	0.4	276
4	6	0.4	271
4	8	0.4	274
6	1	0.4	293
6	2	0.4	286
6	4	0.4	296
6	6	0.4	306
6	8	0.4	301
8	1	0.4	307
8	2	0.4	231
8	4	0.4	318
8	6	0.4	240
8	8	0.4	238
1	1	0.6	247
1	2	0.6	241
1	4	0.6	245
1	6	0.6	243
1	8	0.6	246
2	1	0.6	255
2	2	0.6	249
2	4	0.6	257
2	6	0.6	266
2	8	0.6	263
4	1	0.6	266
4	2	0.6	276
4	4	0.6	277
4	6	0.6	274
4	8	0.6	286
6	1	0.6	286
6	2	0.6	282
6	4	0.6	294
6	6	0.6	290
6	8	0.6	290
8	1	0.6	246
8	2	0.6	309
8	4	0.6	345
8	6	0.6	547
8	8	0.6	260
1	1	0.8	254
1	2	0.8	255
1	4	0.8	269
1	6	0.8	261
1	8	0.8	259
2	1	0.8	245
2	2	0.8	248
2	4	0.8	260
2	6	0.8	268
2	8	0.8	257
4	1	0.8	288
4	2	0.8	293

Continua na próxima página

Tabela 7 – Continua na próxima página

Número de *mappers*	Número de *reducers*	Taxa de *slowstart*	Tempo de execução (s)
4	4	0.8	302
4	6	0.8	298
4	8	0.8	272
6	1	0.8	303
6	2	0.8	301
6	4	0.8	293
6	6	0.8	315
6	8	0.8	310
8	1	0.8	227
8	2	0.8	245
8	4	0.8	310
8	6	0.8	249
8	8	0.8	307

Ao executar o Hadoop repetidas vezes com configurações distintas, foi criado um conjunto de treinamento no intuito de ser a entrada para a criação dos modelos de Função de base radial (RBF), Regressão Linear (LR), Máquinas de Vetores de Suporte (SVM), Processos Gaussianos (GP) e Perceptron de Múltiplas Camadas (MLP) no Weka para prever o tempo de execução para as configurações pertencente ao intervalo pertencente ao espaço de busca especificado no Terasort.

Tabela 8 – Conjunto de treinamento obtido para Terasort.

Número de *mappers*	Número de *reducers*	Taxa de *slowstart*	Tempo de execução (s)
1	1	0.05	278
1	2	0.05	240
1	4	0.05	268
1	6	0.05	242
1	8	0.05	292
2	1	0.05	242
2	2	0.05	226
2	4	0.05	280
2	6	0.05	228
2	8	0.05	292
4	1	0.05	240
4	2	0.05	290
4	4	0.05	291
4	6	0.05	279
4	8	0.05	301
6	1	0.05	231
6	2	0.05	236
6	4	0.05	240
6	6	0.05	249
6	8	0.05	250
8	1	0.05	235
8	2	0.05	220
8	4	0.05	230
8	6	0.05	304
8	8	0.05	290

Continua na próxima página

Tabela 8 – Continua na próxima página

Número de *mappers*	**Número de *reducers***	**Taxa de *slowstart***	**Tempo de execução (s)**
1	1	0.2	274
1	2	0.2	232
1	4	0.2	283
1	6	0.2	276
1	8	0.2	275
2	1	0.2	273
2	2	0.2	220
2	4	0.2	289
2	6	0.2	238
2	8	0.2	308
4	1	0.2	239
4	2	0.2	300
4	4	0.2	283
4	6	0.2	278
4	8	0.2	309
6	1	0.2	234
6	2	0.2	224
6	4	0.2	237
6	6	0.2	239
6	8	0.2	290
8	1	0.2	246
8	2	0.2	273
8	4	0.2	281
8	6	0.2	539
8	8	0.2	293
1	1	0.4	246
1	2	0.4	237
1	4	0.4	281
1	6	0.4	244
1	8	0.4	243
2	1	0.4	230
2	2	0.4	281
2	4	0.4	276
2	6	0.4	287
2	8	0.4	283
4	1	0.4	238
4	2	0.4	298
4	4	0.4	298
4	6	0.4	237
4	8	0.4	300
6	1	0.4	233
6	2	0.4	232
6	4	0.4	221
6	6	0.4	254
6	8	0.4	245
8	1	0.4	244
8	2	0.4	297
8	4	0.4	241
8	6	0.4	288
8	8	0.4	286
1	1	0.6	253
1	2	0.6	277
1	4	0.6	298
1	6	0.6	301
1	8	0.6	249

Continua na próxima página

Tabela 8 – Continua na próxima página

Número de *mappers*	**Número de *reducers***	**Taxa de *slowstart***	**Tempo de execução (s)**
2	1	0.6	281
2	2	0.6	296
2	4	0.6	281
2	6	0.6	296
2	8	0.6	251
4	1	0.6	683
4	2	0.6	500
4	4	0.6	319
4	6	0.6	287
4	8	0.6	322
6	1	0.6	265
6	2	0.6	247
6	4	0.6	248
6	6	0.6	266
6	8	0.6	273
8	1	0.6	265
8	2	0.6	304
8	4	0.6	262
8	6	0.6	310
8	8	0.6	267
1	1	0.8	266
1	2	0.8	264
1	4	0.8	269
1	6	0.8	301
1	8	0.8	305
2	1	0.8	278
2	2	0.8	256
2	4	0.8	313
2	6	0.8	320
2	8	0.8	285
4	1	0.8	329
4	2	0.8	308
4	4	0.8	307
4	6	0.8	255
4	8	0.8	276
6	1	0.8	263
6	2	0.8	229
6	4	0.8	242
6	6	0.8	254
6	8	0.8	254
8	1	0.8	253
8	2	0.8	245
8	4	0.8	312
8	6	0.8	310
8	8	0.8	273